Last updated: November 1, 2026

🗺️ Four-year roadmap · Year 1, Spring

About this course

This free course has 9 units and 4 labs. Those are the real counts, not a longer syllabus cut short.

A check is one number or code result on a course page that a script recomputes before the page is published. The printed number and the recomputed number have to match.

> This is a free self-study course outline written by Saving Planets. It is not a university course, it carries no academic credit, and no school is involved.
>
> Its scope follows first-year programming courses such as MIT 6.100L (formerly 6.0001), Harvard's CS50, Stanford's CS106A and Georgia Tech's CS 1371 for engineers. We link to free materials throughout. None of these organizations endorse this page.
>
> What we add: every example is real Python that our build script runs before the page goes live, and its printed output must match what you see here exactly. The data comes from real public sources: NOAA's Mauna Loa CO_2 record, NASA's GISTEMP temperature analysis and USGS river gauges.
>
> You need nothing installed: use Google Colab or JupyterLite, which runs Python inside your browser with Pyodide. Your progress is saved only in this browser.

How this course helps save the planet

Climate and environmental science runs on code. The CO_2 curve, global temperature records and river forecasts are all built by programs reading public data, and you can read the same files.

Each unit ends with a short, sourced section showing how its ideas help people and the planet.

For instructors

You can teach this course, or let a student work through it alone. It is free. Saving Planets does not give a grade or school credit.

- This page is the whole course: 9 units and 4 labs (ENGR 130).
- Plan on 15 weeks, about 6 to 9 hours a week, including the lab weeks.

- Follow the 15-week schedule. Read the unit, do the practice on paper, then open the answer. In a lab week, do the lab or the home version.
- A check is one number or code result on the page that a script recomputes before we publish. The number you see and the recomputed number have to match.
- Use Print this page, or download the PDF study guide. The formula sheet is near the end.
- Ask: how does this unit help people and the planet? Each unit has a green box with a sourced fact.
- Copy the teacher guide template and write in your class name, the time you have, and the questions you will ask. Open the teacher guide template

Before you start

Prerequisites

- High-school algebra.
- No programming experience needed.
- Physics I and Calculus I help for the simulation unit but are not required.

How to study

- Type every example yourself instead of pasting; your fingers learn syntax.
- Predict the output before you run the code.
- When stuck, read the error message from the bottom up.
- Change one thing at a time and rerun.
- Do every practice problem before opening the answer.
- Retake the quizzes a few days later. Spaced practice beats one long session.

What you need

- A computer or tablet with a modern web browser.
- A free Google account for Colab, or JupyterLite with no account at all.
- Optional: Python 3 installed locally with NumPy and Matplotlib.

Suggested 15-week schedule

About 6 to 9 hours a week: reading, practice problems and, in lab weeks, the lab.

Week	Unit	Topics	Reading
1	Getting Started: Variables, Types & Expressions	Notebooks, variables, types, expressions	Python tutorial §3
2	Control Flow: Conditions & Loops	Conditions and loops	Python tutorial §4

Week	Unit	Topics	Reading
3	Functions	Functions	MIT 6.100L
4	Lists, Dictionaries & Strings	Lists, dictionaries, strings	Python tutorial §5
5	Files & CSV Data	Files and CSV	csv module docs
6	Files & CSV Data	Real data with errors; Lab A	NOAA GML
7	NumPy & Matplotlib	NumPy arrays; midterm practice	NumPy basics
8	NumPy & Matplotlib	Matplotlib; Lab B	Matplotlib pyplot
9	Debugging & Testing	Debugging and testing; Lab C	pytest docs
10	Algorithms & Complexity	Search and complexity	MIT 6.100L
11	Algorithms & Complexity	Bisection and numerical methods	MIT 6.100L
12	Numerical Simulation	Projectile simulation	Physics I
13	Numerical Simulation	Orbits and populations	Calculus II
14	Numerical Simulation	Drag and convergence; Lab D	Intro to Aerospace
15	Review	Review and final practice set	All units

Your progress

Tick units and labs as you finish them, and take the quizzes. Progress and quiz scores are saved only in this browser, never sent to us.

Units

Getting Started: Variables, Types & Expressions

Run your first Python in a browser notebook, store values in variables, and compute with numbers and text.

Learning objectives

By the end of this unit you can:

- Open a Colab or JupyterLite notebook and run cells.
- Use variables and the types int, float, str and bool.
- Use arithmetic operators, including //, % and **.
- Format output with f-strings.

Key concepts

- A program is a sequence of instructions. A notebook lets you run them one cell at a time and see results immediately.
- A variable is a name bound to a value: mass_kg = 1100. Choose names that say what and in which units.
- Every value has a type. int (whole numbers), float (decimals), str (text) and bool (True/False) are the basics; type() tells you which.
- / always gives a float; // is floor division; % is the remainder; ** is a power.
- Floats are binary approximations: 0.1 + 0.2 is not exactly 0.3, so compare floats with a tolerance.

Key equations

$$W = mg$$

`W=mg`

Example 1.1 computes a weight

$$\text{change} = \frac{x_{new} - x_{old}}{x_{old}}$$

`change=_new-x_oldx_old`

Relative change (Example 1.2)

Worked examples, step by step

Example 1.1 Compute the weight of a 1,100 kg aircraft and check the types.

```
mass_kg = 1100
g = 9.81
weight = mass_kg * g
print(weight)
print(type(mass_kg), type(weight))
```

1. mass_kg is an int; multiplying by the float g gives a float.

Output

```
10791.0
<class 'int'> <class 'float'>
```

Answer: 10,791 N, a float.

Example 1.2 How much has Mauna Loa CO_2 risen from the 1959 to the 2025 annual mean (NOAA)?

```
ppm_1959 = 315.98
ppm_2025 = 427.35
rise = ppm_2025 - ppm_1959
print(f"CO2 rose {rise:.2f} ppm, or {rise / ppm_1959:.1%}")
```

1. Subtract, then divide by the starting value.
2. The f-string formats :.2f (2 decimals) and :.1% (percent).

Output

```
CO2 rose 111.37 ppm, or 35.2%
```

Answer: 111.37 ppm, a 35.2% rise.

Example 1.3 Explore operators and float surprises.

```
print(7 / 2, 7 // 2, 7 % 2, 2 ** 10)
print(0.1 + 0.2 == 0.3, abs((0.1 + 0.2) - 0.3) < 1e-9)
print(int("42") + 1, str(42) + "1")
```

1. 7/2 = 3.5, 7//2 = 3, 7%2 = 1.
2. 0.1 + 0.2 == 0.3 is False; a tolerance test is True.
3. int("42") converts text to a number; str(42) + "1" joins text.

Output

```
3.5 3 1 1024
False True
43 421
```

Answer: Use tolerances for floats and convert types on purpose.

Practice problems

Work each on paper first, then check.

1. What does 17 % 5 give?

   2.

2. What type is 3 * 2.0?

   float (6.0).

3. Write an f-string that prints pi to 3 decimals.

   f"{math.pi:.3f}" → 3.142 (after import math).

4. Why is mass_kg a better name than m?

   It states the quantity and its unit, preventing unit mistakes.

Common misconceptions

Myth: = means equals.
Actually: In Python = assigns; == compares.

Myth: 0.1 + 0.2 == 0.3 is True.
Actually: It is False because of binary rounding.

Myth: "5" + 5 gives 10.
Actually: It is a TypeError; convert with int("5").

Flashcards

Variable	**int / float**
Name bound to a value.	Whole number / decimal number.

str

Text.

bool

True or False.

f-string

f"{x:.2f}" formatting.

//, %

Floor division, remainder.

Self-check quiz

1. 7 // 2 =

3.5

3

4

1

3. Floor division.

2. type(2.0) is...

int

float

str

bool

float. It has a decimal point.

3. 2 ** 10 =

20

100

1024

12

1024. Power.

4. Assignment in Python uses...

==

=

:=only

->

=. = binds a name.

Try this quiz again

How this helps save the planet

The numbers in Example 1.2 come straight from NOAA's Global Monitoring Laboratory, which publishes the Mauna Loa CO_2 record as a free CSV file. A few lines of Python turn it into the headline facts of climate science.

On Saving Planets: Calculus I

Sources: NOAA GML: Trends in atmospheric CO_2, data (https://gml.noaa.gov/ccgg/trends/data.html) · Google Colab (free notebooks) (https://colab.research.google.com/)

Free learning resources

- The official Python tutorial (https://docs.python.org/3/tutorial/): Sections 3 and 4.
- Google Colab (free notebooks) (https://colab.research.google.com/): Start a notebook.
- JupyterLite: Python in your browser (Pyodide) (https://jupyter.org/try-jupyter/lab/): No-account option.

☐ **I finished this unit**

Control Flow: Conditions & Loops

Make programs decide with if/elif/else and repeat with for and while loops.

Learning objectives

By the end of this unit you can:

- Write if/elif/else with comparison and boolean operators.
- Loop over ranges and sequences with for.
- Use while loops with a clear stopping condition.
- Accumulate totals and counts.

Key concepts

- Indentation defines blocks in Python. Four spaces per level is standard.
- Comparisons (<, >=, ==, !=) produce booleans; combine with and, or, not.
- for x in sequence repeats once per item; range(a, b) gives a, a+1, ..., b−1.
- while repeats until its condition is False. Make sure something in the loop changes it, or it never ends.
- The accumulator pattern: start a total at 0 and add inside the loop.

Key equations

$$\sum_{n=1}^{100} n = \frac{100 \cdot 101}{2} = 5050$$

_n=1^100n=(100·101)/(2)=5050

Gauss's sum checks Example 2.2

$$x_{k+1} = x_k + \Delta$$

x_k+1=x_k+Δ

A loop that steps a value

Worked examples, step by step

Example 2.1 Classify NASA's 2024 global temperature anomaly.

```
anomaly = 1.28   # NASA GISTEMP, 2024, deg C vs 1951-1980
if anomaly >= 1.0:
    print("at least 1.0 C above the 1951-1980 average")
```

```
elif anomaly > 0:
    print("warmer than average")
else:
    print("at or below average")
```

1. 1.28 ≥ 1.0, so the first branch runs and the rest are skipped.

Output

```
at least 1.0 C above the 1951-1980 average
```

Answer: Branches are tested in order; only the first true one runs.

Example 2.2 Add the numbers 1 to 100 with a loop.

```
total = 0
for n in range(1, 101):
    total += n
print(total)
```

1. range(1, 101) stops before 101.
2. total += n adds each number.

Output

```
5050
```

Answer: 5,050, matching Gauss's formula.

Example 2.3 If CO_2 kept rising 2.5 ppm a year from the 2025 mean, when would it pass 450 ppm?

```
ppm = 427.35   # 2025 annual mean, Mauna Loa
year = 2025
while ppm < 450:
    ppm += 2.5
    year += 1
print(year, round(ppm, 2))
```

1. The loop adds 2.5 ppm and one year until ppm ≥ 450.

Output

```
2035 452.35
```

Answer: In 2035, at 452.35 ppm under this simple constant-rate scenario (not a forecast).

Practice problems

Work each on paper first, then check.

1. How many times does for i in range(3, 10, 2) run?

   4 times (3, 5, 7, 9).

2. Write a loop that counts the odd numbers below 20.

   count = sum(1 for n in range(20) if n % 2): 10.

3. What is wrong with: while x < 10: print(x)

   x never changes, so it loops forever.

Common misconceptions

Myth: range(1, 10) includes 10.
Actually: It stops at 9.

Myth: else after if is required.
Actually: It is optional.

Myth: Indentation is just style.
Actually: In Python it defines the program's structure.

Flashcards

if / elif / else

Choose one branch.

for

Repeat per item.

while

Repeat while true.

range(a, b, s)

a up to b−1, step s.

Accumulator

break / continue

total += x in a loop.

Leave loop / skip to next pass.

Self-check quiz

1. list(range(2, 6)) is...

[2, 3, 4, 5, 6]

[2, 3, 4, 5]

[3, 4, 5, 6]

[2, 6]

[2, 3, 4, 5]. Stops before 6.

2. Which keyword adds another test after if?

else if

elif

then

case

elif. elif.

3. A while loop stops when...

its condition is False

after 10 times

never

at a print

its condition is False. Condition checked each pass.

4. not (True and False) is...

True

False

None

an error

True. True and False is False.

Try this quiz again

How this helps save the planet

Thresholds like 1.0 °C or 450 ppm are exactly what monitoring programs test with if statements, flagging records and alerts automatically as new data arrives.

On Saving Planets: Physics I

Sources: NASA GISS: GISTEMP v4 surface temperature analysis (https://data.giss.nasa.gov/gistemp/) · NOAA GML: Trends in atmospheric CO_2, data (https://gml.noaa.gov/ccgg/trends/data.html)

Free learning resources

- The official Python tutorial (https://docs.python.org/3/tutorial/): Section 4: control flow.
- MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/): Lectures on branching and iteration.

☐ **I finished this unit**

Functions

Package logic into reusable, testable functions with parameters, defaults and return values.

Learning objectives

By the end of this unit you can:

- Define functions with def, parameters and return.
- Use default and keyword arguments.
- Write docstrings.
- Understand local scope.

Key concepts

- A function names a piece of logic. Write it once, test it once, use it everywhere.
- Parameters receive values; return sends a result back. A function with no return gives None.
- Default arguments (cl_max=1.6) make common cases short; keyword arguments make calls readable.
- Variables created inside a function are local and vanish when it returns.
- A docstring (the string on the first line) documents what the function does.

Key equations

$$F = \tfrac{9}{5}C + 32$$

F=95C+32

Example 3.1

$$V_s = \sqrt{\frac{2W}{\rho S C_{L,\max}}}$$

V_s=2Wρ SC_L,

Example 3.2, from Intro to Aerospace

Worked examples, step by step

Example 3.1 Write and test a Celsius-to-Fahrenheit function.

```
def c_to_f(c):
    """Convert Celsius to Fahrenheit."""
    return c * 9 / 5 + 32
```

```
for c in (-40, 0, 1.28, 100):
    print(c, "->", round(c_to_f(c), 2))
```

1. −40 is the same on both scales, a handy check.
2. round(..., 2) tidies the floats.

Output

```
-40 -> -40.0
0 -> 32.0
1.28 -> 34.3
100 -> 212.0
```

Answer: A 1.28 °C anomaly is a 2.3 °F difference, but 1.28 °C as a temperature is 34.3 °F; units matter.

Example 3.2 Turn the stall-speed formula into a function and use it at two densities.

```
def stall_speed(weight_n, rho, area, cl_max=1.6):
    return (2 * weight_n / (rho * area * cl_max)) ** 0.5

print(round(stall_speed(10791, 1.227, 16.2), 1))
print(round(stall_speed(10791, 0.738, 16.2), 1))
```

1. cl_max has a default, so callers can omit it.
2. ** 0.5 is a square root.

Output

```
26.0
33.6
```

Answer: 26.0 m/s at sea level and 33.6 m/s at 5,000 m, matching the aerospace course.

Example 3.3 Write a function that returns year-to-year changes of a list (Mauna Loa annual means 2022–2025).

```
def growth(values):
    """Year-to-year changes of a list."""
    return [b - a for a, b in zip(values, values[1:])]
```

```
co2 = [418.53, 421.08, 424.61, 427.35]   # 2022-2025
print([round(d, 2) for d in growth(co2)])
```

1. zip pairs each value with the next.
2. A list comprehension builds the result.

Output

```
[2.55, 3.53, 2.74]
```

Answer: Increases of 2.55, 3.53 and 2.74 ppm.

Practice problems

Work each on paper first, then check.

1. Write area_circle(r) and test it at r = 2.

   return math.pi * r**2 → 12.566....

2. What does a function without return give?

   None.

3. Why test c_to_f at −40?

   Both scales agree there, so the expected answer is known.

Common misconceptions

Myth: print and return are the same.
Actually: print shows a value; return hands it back for further use.

Myth: Changing a parameter changes the caller's variable.
Actually: Rebinding a parameter is local (mutable objects are an exception).

Myth: Functions should be long and do everything.
Actually: Small, single-purpose functions are easier to test.

Flashcards

def	**return**
Defines a function.	Sends a value back.

Parameter

Name for an input.

Default argument

Value used if omitted.

Docstring

Documentation string.

Scope

Where a name is visible.

Self-check quiz

1. Define a function with...

func

def

function

lambda only

def. def name(params):

2. A function with no return statement returns...

0

None

False

an error

None. Implicit None.

3. In def f(x, k=2), k is...

required

a default argument

global

a constant

a default argument. Has a default.

4. Variables made inside a function are...

global

local

shared

permanent

local. Local scope.

Try this quiz again

How this helps save the planet

Climate and hydrology codes are built from small, tested functions like these, for unit conversions, rates of change and physical formulas, which is how large models stay trustworthy.

On Saving Planets: Intro to Aerospace Engineering

Sources: NOAA GML: Trends in atmospheric CO_2, data (https://gml.noaa.gov/ccgg/trends/data.html) · The official Python tutorial (https://docs.python.org/3/tutorial/)

Free learning resources

- The official Python tutorial (https://docs.python.org/3/tutorial/): Section 4.8: defining functions.
- Harvard CS50's Introduction to Programming with Python (https://cs50.harvard.edu/python/): Functions and variables.

☐ **I finished this unit**

Lists, Dictionaries & Strings

Store collections of data: lists for sequences, dictionaries for lookups, and strings for text.

Learning objectives

By the end of this unit you can:

- Index, slice and modify lists.
- Use len, sum, min, max and sorted.
- Build and query dictionaries.
- Split strings and count words.

Key concepts

- A list holds an ordered sequence: flows[0] is the first item, flows[-1] the last, flows[2:5] a slice.
- Lists are mutable: append, insert, remove and sort change them in place.
- A dictionary maps keys to values: gistemp[2024] → 1.28. Lookups are fast no matter how big it is.
- List comprehensions build lists concisely: [x * 2 for x in xs if x > 0].
- Strings are sequences of characters with methods like split, strip, lower and startswith.

Key equations

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```
x=1n_i=1^n x_i
```

Mean of a list (Example 4.1)

Worked examples, step by step

Example 4.1 Summarize ten days of USGS daily mean flow for the Potomac at Little Falls (June 1–10, 2025, in cubic feet per second).

```
flows = [44700, 40200, 29100, 22700, 17300, 14100, 12600, 12200, 14300, 22600]
print(len(flows), max(flows), min(flows))
print(sum(flows) / len(flows))
print(sorted(flows)[:3])
```

1. len, max and min work on any list.
2. sum/len is the mean; sorted returns a new list.

Output

```
10 44700 12200
22980.0
[12200, 12600, 14100]
```

Answer: Mean 22,980 cfs; range 12,200–44,700 cfs.

Example 4.2 Find the warmest year in a dictionary of NASA GISTEMP anomalies.

```
gistemp = {2016: 1.01, 2020: 1.01, 2023: 1.17, 2024: 1.28, 2025: 1.18}
warmest = max(gistemp, key=gistemp.get)
print(warmest, gistemp[warmest])
print([y for y, a in gistemp.items() if a > 1.1])
```

1. max with key=gistemp.get compares values.
2. A comprehension filters items.

Output

```
2024 1.28
[2023, 2024, 2025]
```

Answer: 2024, at 1.28 °C; 2023–2025 are all above 1.1 °C.

Example 4.3 Count words with a dictionary.

```
words = "reduce reuse recycle reduce repair reduce".split()
counts = {}
for w in words:
    counts[w] = counts.get(w, 0) + 1
print(counts)
```

1. get(w, 0) returns 0 for a new word.

Output

```
{'reduce': 3, 'reuse': 1, 'recycle': 1, 'repair': 1}
```

Answer: “reduce” appears 3 times.

Practice problems

Work each on paper first, then check.

1. What is [10, 20, 30, 40][1:3]?

   [20, 30].

2. How do you add 2026: 1.2 to gistemp?

   gistemp[2026] = 1.2.

3. Square each number in xs with a comprehension.

   [x**2 for x in xs].

Common misconceptions

Myth: Lists start at index 1.
Actually: They start at 0.

Myth: sorted(xs) changes xs.
Actually: sorted returns a new list; xs.sort() changes it.

Myth: Dictionary keys can repeat.
Actually: Each key is unique; assigning again replaces the value.

Flashcards

List

Ordered, mutable sequence.

Index / slice

xs[i], xs[a:b].

Dictionary

Key → value map.

Comprehension

[f(x) for x in xs if cond].

Tuple

Immutable sequence.

str.split

Text → list of pieces.

Self-check quiz

1. xs[-1] is...

the first item

the last item

an error

the length

the last item. Negative indices count from the end.

2. len({"a": 1, "b": 2}) =

1

2

4

3

2. Two keys.

3. "a,b,c".split(",") gives...

"abc"

["a", "b", "c"]

("a","b","c")

3

["a", "b", "c"]. A list of strings.

4. Lists are...

immutable

mutable

dictionaries

always sorted

mutable. They can change in place.

Try this quiz again

How this helps save the planet

USGS publishes daily streamflow for thousands of river gauges. The ten numbers in Example 4.1 are real: a high-flow pulse on the Potomac receding over a week, the kind of record used for water supply and flood planning.

On Saving Planets: Rivers hub

Sources: USGS: Potomac River near Washington, DC (Little Falls), site 01646500 (https://waterdata.usgs.gov/monitoring-location/USGS-01646500/) · NASA GISS: GISTEMP v4 surface temperature analysis (https://data.giss.nasa.gov/gistemp/)

Free learning resources

- The official Python tutorial (https://docs.python.org/3/tutorial/): Section 5: data structures.
- USGS Water Services (data API) (https://waterservices.usgs.gov/): How to request USGS data.

☐ **I finished this unit**

Files & CSV Data

Read and write files, parse CSV data, and handle messy rows without crashing.

Learning objectives

By the end of this unit you can:

- Open, read and write text files with with.
- Parse CSV with the csv module.
- Convert strings to numbers and handle errors with try/except.
- Download a real dataset and analyze it.

Key concepts

- with open(name) as f: opens a file and closes it automatically.
- CSV (comma-separated values) is the most common format for public data. csv.DictReader turns each row into a dictionary keyed by the header.
- Everything read from a file is text: convert with int() and float().
- Real data has comments, blanks and bad values. try/except ValueError lets you skip or log them.
- NOAA’s Mauna Loa annual-mean file starts with # comment lines, then year,mean,unc.

Key equations

$$\text{rate} = \frac{x_{last} - x_{first}}{t_{last} - t_{first}}$$

```
rate=_last-x_firstt_last-t_first
```

Average rate of change (Example 5.1)

$$1 \text{ cfs} = 0.0283168 \text{ m}^3/\text{s}$$

```
1\ cfs=0.0283168\ m^3/s
```

Unit conversion (Example 5.2)

Worked examples, step by step

Example 5.1 Parse the last five rows of NOAA’s Mauna Loa annual-mean CSV and find the average rate of rise.

```
import csv, io
text = """year,mean,unc
2021,416.41,0.12
```

```
2022,418.53,0.12
2023,421.08,0.12
2024,424.61,0.12
2025,427.35,0.12
"""
rows = list(csv.DictReader(io.StringIO(text)))
years = [int(r["year"]) for r in rows]
ppm = [float(r["mean"]) for r in rows]
print(years[0], years[-1], ppm[-1])
print(round((ppm[-1] - ppm[0]) / (years[-1] - years[0]), 2), "ppm per year")
```

1. io.StringIO lets csv read a string as if it were a file.
2. Convert the text fields to int and float.

Output

```
2021 2025 427.35
2.73 ppm per year
```

Answer: 2.73 ppm per year from 2021 to 2025.

Example 5.2 Write a small CSV of USGS flows, read it back and convert to m³/s.

```
with open("flows.csv", "w") as f:
    f.write("date,cfs\n2025-06-01,44700\n2025-06-02,40200\n2025-06-03,29100\n")
with open("flows.csv") as f:
    next(f)                      # skip the header
    for line in f:
        date, cfs = line.strip().split(",")
        print(date, round(int(cfs) * 0.0283168, 1), "m3/s")
```

1. next(f) skips the header line.
2. strip removes the newline; split separates fields.

Output

```
2025-06-01 1265.8 m3/s
2025-06-02 1138.3 m3/s
2025-06-03 824.0 m3/s
```

Answer: 44,700 cfs ≈ 1,265.8 m³/s.

Example 5.3 Read lines with comments and bad values without crashing.

```
lines = ["# comment", "year,mean", "2024,424.61", "2025,bad", "2026,"]
good = []
for line in lines:
    if line.startswith("#") or line.startswith("year"):
        continue
    year, value = line.split(",")
    try:
        good.append((int(year), float(value)))
    except ValueError:
        print("skipping", repr(line))
print(good)
```

1. continue skips comments and the header.
2. float("bad") and float("") raise ValueError, which is caught.

Output

```
skipping '2025,bad'
skipping '2026,'
[(2024, 424.61)]
```

Answer: Only the valid row is kept; the bad ones are reported.

Practice problems

Work each on paper first, then check.

1. How do you read the full NOAA file in Colab?

   Use urllib.request.urlopen(url) or download it, then skip lines starting with #.

2. What does csv.DictReader give for each row?

   A dict keyed by column name, with string values.

3. Why use with open(...)?

   The file is closed automatically, even if an error occurs.

Common misconceptions

Myth: Numbers in a CSV are numbers.

Actually: They are read as strings until converted.

Myth: Bad rows mean the dataset is useless.

Actually: Skip and log them, then report how many.

Myth: Opening a file in "w" mode adds to it.

Actually: "w" overwrites; "a" appends.

Flashcards

with open()

Opens and auto-closes a file.

CSV

Comma-separated values.

csv.DictReader

Rows as dicts.

try / except

Handle errors.

ValueError

Bad conversion.

Mode "w" / "a"

Overwrite / append.

Self-check quiz

1. float("3.5") gives...

"3.5"

3.5

3

an error

3.5. Converts text to a float.

2. float("") raises...

TypeError

ValueError

KeyError

nothing

ValueError. Empty string is not a number.

3. The "a" file mode...

overwrites

appends

reads

deletes

appends. Append.

4. csv.DictReader keys come from...

row numbers

the header row

the filename

random

the header row. First line.

Try this quiz again

How this helps save the planet

NOAA, NASA and USGS share their measurements as free CSV files. Being able to read them yourself means you can check headlines against the source data.

On Saving Planets: Calculus II

Sources: NOAA GML: Mauna Loa CO_2 annual means (CSV) (https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_annmean_mlo.csv) · Python docs: the csv module (https://docs.python.org/3/library/csv.html)

Free learning resources

- Python docs: the csv module (https://docs.python.org/3/library/csv.html): Reading and writing CSV.
- NOAA GML: Mauna Loa CO_2 annual means (CSV) (https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_annmean_mlo.csv): The real file.
- The official Python tutorial (https://docs.python.org/3/tutorial/): Section 7.2: files.

☐ **I finished this unit**

NumPy & Matplotlib

Compute on whole arrays at once with NumPy, fit trends, and make clear plots with Matplotlib.

Learning objectives

By the end of this unit you can:

- Create NumPy arrays and use vectorized arithmetic.
- Use mean, diff, boolean masks and polyfit.
- Plot data with labeled axes and titles.
- Save figures to files.

Key concepts

- A NumPy array holds numbers of one type and does math on all of them at once: cfs * 0.0283168 converts every value.
- np.diff gives successive differences; a boolean mask like cfs > 20000 selects or counts values.
- np.polyfit(x, y, 1) fits a straight line and returns slope and intercept.
- Matplotlib's pyplot draws figures: fig, ax = plt.subplots(); ax.plot(x, y); label axes with units; fig.savefig saves the image.
- A good plot has labeled axes with units, a title and a cited data source.

Key equations

$$y \approx mx + b$$

Least-squares line from np.polyfit

$$V = \sum Q_i \, \Delta t$$

`V= Q_i Δ t`

Volume from daily flows (Example 6.2)

Worked examples, step by step

Example 6.1 Use NumPy on Mauna Loa annual means 2021–2025: differences, mean and trend.

```
import numpy as np
years = np.array([2021, 2022, 2023, 2024, 2025])
ppm = np.array([416.41, 418.53, 421.08, 424.61, 427.35])
print(np.diff(ppm))
print(round(ppm.mean(), 3), round(np.diff(ppm).mean(), 3))
```

```
slope, intercept = np.polyfit(years, ppm, 1)
print(round(slope, 3))
```

1. np.diff gives four yearly increases.
2. polyfit's slope is the best-fit rate.

Output

```
[2.12 2.55 3.53 2.74]
421.596 2.735
2.796
```

Answer: Best-fit rise ≈ 2.80 ppm per year over 2021–2025.

Example 6.2 Convert ten days of Potomac flow to m^3/s, count high-flow days and find the total volume.

```
import numpy as np
cfs = np.array([44700, 40200, 29100, 22700, 17300, 14100, 12600, 12200, 14300, 22600])
m3s = cfs * 0.0283168
print(np.round(m3s[:3], 1))
print(int((cfs > 20000).sum()), "days above 20,000 cfs")
print(round(m3s.sum() * 86400 / 1e9, 3), "km3 in 10 days")
```

1. Multiply the whole array at once.
2. (cfs > 20000).sum() counts True values.
3. Sum of m^3/s × 86,400 s per day.

Output

```
[1265.8 1138.3  824. ]
5 days above 20,000 cfs
0.562 km3 in 10 days
```

Answer: Five days above 20,000 cfs; about 0.56 km^3 of water passed in ten days.

Example 6.3 Plot selected NASA GISTEMP annual anomalies and save the figure.

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
years = [1880, 1951, 1980, 2000, 2016, 2020, 2023, 2024, 2025]
```

```
anom = [-0.17, -0.07, 0.26, 0.39, 1.01, 1.01, 1.17, 1.28, 1.18]
fig, ax = plt.subplots()
ax.plot(years, anom, "o-")
ax.set_xlabel("Year"); ax.set_ylabel("Anomaly (C)")
ax.set_title("NASA GISTEMP v4, selected years")
fig.savefig("gistemp.png")
print(len(ax.lines[0].get_xdata()), "points saved")
```

1. matplotlib.use("Agg") lets code save images without a screen (Colab shows plots inline anyway).
2. Label axes and title; save with savefig.

Output

```
9 points saved
```

Answer: A saved line plot of 9 points; open gistemp.png to see it.

Practice problems

Work each on paper first, then check.

1. np.array([1, 2, 3]) * 2 gives?

   array([2, 4, 6]).

2. How do you select values above 1.0 in an array a?

   a[a > 1.0].

3. Which call adds an axis label?

   ax.set_xlabel("Year").

Common misconceptions

Myth: NumPy arrays and lists add the same way.
Actually: [1, 2] + [3, 4] concatenates; arrays add elementwise.

Myth: A fitted line proves a cause.
Actually: It summarizes a trend; causes need physical understanding.

Myth: Plots without units are fine.
Actually: Always label units and cite the source.

Flashcards

np.array

Fast numeric array.

Vectorized

Math on whole arrays.

Boolean mask

a[a > x].

np.polyfit

Least-squares polynomial fit.

plt.subplots

Make a figure and axes.

savefig

Save a plot.

Self-check quiz

1. np.diff([1, 4, 9]) =

[3, 5]

[1, 4, 9]

[0, 3, 5]

14

[3, 5]. Successive differences.

2. np.polyfit(x, y, 1) returns...

the mean

slope and intercept

a plot

the max

slope and intercept. Degree-1 fit.

3. (a > 0).sum() counts...

positive values

all values

zeros

nothing

positive values. True counts as 1.

4. fig.savefig("x.png")...

prints

saves the figure

deletes it

shows code

saves the figure. Writes an image file.

Try this quiz again

How this helps save the planet

NASA's GISTEMP analysis is published with its data so anyone can reproduce the global temperature record. Plotting it yourself, with labeled units and the source, is the honest way to share climate facts.

On Saving Planets: Climate data on this site: CO2M

Sources: NASA GISS: GISTEMP v4 surface temperature analysis (https://data.giss.nasa.gov/gistemp/) · NumPy: the absolute basics for beginners (https://numpy.org/doc/stable/user/absolute_beginners.html)

Free learning resources

- NumPy: the absolute basics for beginners (https://numpy.org/doc/stable/user/absolute_beginners.html): Arrays for beginners.
- Matplotlib: pyplot tutorial (https://matplotlib.org/stable/tutorials/pyplot.html): Plotting basics.
- NASA GISTEMP v4 global means (CSV) (https://data.giss.nasa.gov/gistemp/tabledata_v4/GLB.Ts+dSST.csv): The real data.

- [ ] **I finished this unit**

Debugging & Testing

Find bugs systematically, read tracebacks, and write tests that catch mistakes before they matter.

Learning objectives

By the end of this unit you can:

- Read a traceback and locate the failing line.
- Use print and assertions to narrow down bugs.
- Raise clear errors for bad input.
- Write unit tests with assert and pytest.

Key concepts

- Read error messages from the bottom: the last line names the error type, the lines above show where it happened.
- Debug scientifically: form a hypothesis, test it with a small input, change one thing.
- Edge cases (empty lists, zero, negative numbers, missing data) cause most bugs.
- An assert states what must be true; a failing assert stops the program at the bug.
- Tests are small functions that check known answers; pytest runs every function named test_... automatically.

Key equations

$$\left|x_{computed} - x_{expected}\right| < \varepsilon$$

Tolerance test for floats

Worked examples, step by step

Example 7.1 Find the bug in a mean function called on an empty list.

```
def mean(xs):
    return sum(xs) / len(xs)

try:
    print(mean([]))
except ZeroDivisionError as e:
    print("bug found:", e)
```

1. Dividing by len([]) = 0 raises ZeroDivisionError.

2. Catching it shows the message.

Output

```
bug found: division by zero
```

Answer: The function needs to handle empty input.

Example 7.2 Fix it with a clear error and add assertion tests.

```
def mean(xs):
    if not xs:
        raise ValueError("mean() of empty list")
    return sum(xs) / len(xs)

assert mean([1, 2, 3]) == 2
assert abs(mean([0.1, 0.2]) - 0.15) < 1e-12
try:
    mean([])
except ValueError:
    print("all tests passed")
```

1. Raise ValueError with a message for empty input.
2. Float results are compared with a tolerance.

Output

```
all tests passed
```

Answer: All tests pass.

Example 7.3 Write a pytest-style test for f_to_c using known points.

```
def f_to_c(f):
    return (f - 32) * 5 / 9

def test_f_to_c():
    assert f_to_c(32) == 0
    assert f_to_c(212) == 100
    assert abs(f_to_c(-40) - (-40)) < 1e-12
```

```
test_f_to_c()
print("ok")
```

1. Known pairs: 32 °F = 0 °C, 212 °F = 100 °C, −40 = −40.
2. In a file, pytest would discover test_f_to_c automatically; here we call it.

Output

```
ok
```

Answer: The function passes all three checks.

Practice problems

Work each on paper first, then check.

1. What does “IndexError: list index out of range” usually mean?

   You used an index ≥ len(list), often an off-by-one error.

2. Write an assert that x is positive.

   assert x > 0, "x must be positive".

3. Name three edge cases for a function that averages data.

   Empty list, a single value, and missing or non-numeric values.

Common misconceptions

Myth: If it runs, it is correct.
Actually: Code can run and give wrong answers; test against known results.

Myth: Compare floats with ==.
Actually: Use a tolerance.

Myth: Tests are only for big projects.
Actually: Small tests save time from the first day.

Flashcards

Traceback	**Edge case**
Error report; read from the bottom.	Unusual input at the boundary.

assert

Stop if a condition is false.

Unit test

Checks one function against known answers.

pytest

Test runner.

raise

Signal an error deliberately.

Self-check quiz

1. The most useful line of a traceback is usually...

the first

the last

the middle

none

the last. It names the error.

2. pytest runs functions named...

run_*

test_*

check_*

main

test_*. Convention.

3. An assert that fails raises...

ValueError

AssertionError

nothing

TypeError

AssertionError. AssertionError.

4. A good first debugging step is...

rewrite everything

reproduce with a small input

add random prints everywhere

ignore it

reproduce with a small input. Shrink the problem.

Try this quiz again

How this helps save the planet

Scientific code that informs policy must be checked. This course practices what it preaches: every code example on this page is run by our build script, and its output must match exactly before the page goes live.

On Saving Planets: Courses

Sources: pytest documentation (https://docs.pytest.org/) · The official Python tutorial (https://docs.python.org/3/tutorial/)

Free learning resources

- pytest documentation (https://docs.pytest.org/): Writing tests.
- The official Python tutorial (https://docs.python.org/3/tutorial/): Section 8: errors and exceptions.

Watch the lecture

Lecture 7 testing debugging exceptions and assertions (https://ocw.mit.edu/courses/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/resources/lecture-7-testing-debugging-exceptions-and-assertions/) (in English)

MIT 6.0001 Introduction to Computer Science and Programming in Python (Fall 2016), MIT OpenCourseWare. License: CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0/) · Terms of use (https://ocw.mit.edu/pages/privacy-and-terms-of-use/). Linked for free study only; Saving Planets is not affiliated with or endorsed by this university.

☐ **I finished this unit**

Algorithms & Complexity

Searching, root finding and how running time grows with input size: big-O thinking.

Learning objectives

By the end of this unit you can:

- Implement linear and binary search.
- Use bisection to solve equations.
- Estimate complexity: O(1), O(log n), O(n), O(n²).
- Choose data structures for speed.

Key concepts

- An algorithm is a precise step-by-step method. Its cost is how the number of steps grows with the input size n.
- Linear search checks items one by one: O(n). Binary search halves a sorted list each step: O(log n).
- Bisection finds a root of f on [a, b] where f changes sign by halving the interval; after n steps the interval is (b − a)/2ⁿ.
- Nested loops over n items are O(n²): fine for 1,000 items, hopeless for a million.
- Dictionary and set lookups are O(1) on average, list searches O(n).

Key equations

$$\text{steps}_{binary} \approx \lceil \log_2 n \rceil$$

Binary search steps

$$n = \left\lceil \log_2 \frac{b-a}{\text{tol}} \right\rceil$$

`n=_2-atol`

Bisection iterations

Worked examples, step by step

Example 8.1 Search a sorted list of one million even numbers for the last one, both ways.

```
def linear_search(xs, target):
    for i, x in enumerate(xs):
        if x == target:
            return i
```

```
    return -1

def binary_search(xs, target):
    lo, hi, steps = 0, len(xs) - 1, 0
    while lo <= hi:
        steps += 1
        mid = (lo + hi) // 2
        if xs[mid] == target:
            return mid, steps
        if xs[mid] < target:
            lo = mid + 1
        else:
            hi = mid - 1
    return -1, steps

data = list(range(0, 2_000_000, 2))
print(linear_search(data, 1_999_998))
print(binary_search(data, 1_999_998))
```

1. Linear search checks 1,000,000 items to reach index 999,999.
2. Binary search needs only 20 steps.

Output

```
999999
(999999, 20)
```

Answer: 20 steps versus a million: $\log_2(10^6) \approx 20$.

Example 8.2 Find $\sqrt{2}$ by bisection on $x^2 - 2$ from $[0, 2]$ to 10^{-6}.

```
def bisect_root(f, a, b, tol=1e-6):
    n = 0
    while b - a > tol:
        m = (a + b) / 2
        if f(a) * f(m) <= 0:
            b = m
        else:
            a = m
        n += 1
    return (a + b) / 2, n

root, n = bisect_root(lambda x: x**2 - 2, 0, 2)
print(round(root, 6), n)
```

1. Each step halves the interval.
2. $2/2^{21} < 10^{-6}$ after 21 steps.

Output

```
1.414214 21
```

Answer: 1.414214 after 21 iterations.

Example 8.3 Compare $\log_2 n$ and n^2 growth.

```
import math
for n in (10, 1_000, 1_000_000):
    print(n, math.ceil(math.log2(n)), n * n)
```

1. math.ceil(math.log2(n)) is the binary-search step count.

Output

```
10 4 100
1000 10 1000000
1000000 20 1000000000000
```

Answer: For $n = 10^6$, $\log_2 n$ is 20 while n^2 is 10^{12}.

Practice problems

Work each on paper first, then check.

1. How many binary-search steps for 1 billion sorted items?

   $\lceil \log_2 10^9 \rceil = 30$.

2. What is the complexity of summing a list?

   O(n).

3. Bisection on [0, 1] to 10^{-3}: how many steps?

   $\lceil \log_2 1{,}000 \rceil = 10$.

Common misconceptions

Myth: Faster computers make complexity irrelevant.

Actually: An $O(n^2)$ method on a million items is 10^{12} steps; no computer fixes that.

Myth: Binary search works on any list.

Actually: Only on sorted data.

Myth: Big-O gives exact time.

Actually: It describes growth, ignoring constants.

Flashcards

Algorithm

Precise step-by-step method.

Big-O

Growth rate of cost.

Linear search

O(n).

Binary search

O(log n), sorted data.

Bisection

Root finding by halving.

$O(n^2)$

Nested loops.

Self-check quiz

1. Binary search is...

O(n)

O(log n)

$O(n^2)$

O(1)

O(log n). Halves each step.

2. Bisection requires...

a derivative

a sign change on [a, b]

sorted data

integers

a sign change on [a, b]. Intermediate value theorem.

3. A loop inside a loop over n items is...

O(n)

$O(n^2)$

O(log n)

O(1)

$O(n^2)$. n × n.

4. Average lookup in a dict is...

O(n)

O(1)

$O(n^2)$

O(log n)

O(1). Hashing.

Try this quiz again

How this helps save the planet

Efficient algorithms make it possible to process the flood of Earth-observation data, from satellites to thousands of river gauges. Choosing O(n log n) over $O(n^2)$ can turn a week of computing into seconds, and less computing also means less energy.

On Saving Planets: Calculus I

Sources: USGS Water Services (data API) (https://waterservices.usgs.gov/) · MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programmi

ng-using-python-fall-2022/)

Free learning resources

- MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/): Lectures on search, bisection and complexity.
- Harvard CS50's Introduction to Programming with Python (https://cs50.harvard.edu/python/): Problem-solving practice.

☐ **I finished this unit**

Numerical Simulation

Step physical models forward in time: a projectile, an orbit and a population, the core of every climate and engineering model.

Learning objectives

By the end of this unit you can:

- Use time-stepping (Euler and Euler-Cromer) to simulate motion.
- Check simulations against exact answers.
- Simulate a circular orbit and a logistic population.
- Explore how step size affects accuracy.

Key concepts

- A simulation updates the state (position, velocity, population) in small time steps dt using the rates of change: x_new = x + v·dt.
- Euler-Cromer updates velocity first, then position with the new velocity; it keeps orbits stable much better than plain Euler.
- Always validate: compare with an exact solution in a case where one exists (no air drag, circular orbit, logistic formula).
- Smaller dt is more accurate but slower; halve dt and see whether the answer changes.
- Weather and climate models are vast versions of these loops, stepping the equations of physics across a global grid.

Key equations

$$v_{k+1} = v_k + a_k\,\Delta t, \quad x_{k+1} = x_k + v_{k+1}\,\Delta t$$

`v_k+1=v_k+a_k Δ t, x_k+1=x_k+v_k+1 Δ t`

Euler-Cromer step

$$R = \frac{v_0^2 \sin 2\theta}{g}$$

`R=(v_0^22θ)/(g)`

Projectile range without drag

$$\vec{a} = -\frac{GM}{r^3}\vec{r}$$

`a=-(GM)/(r^3) r`

Gravity for an orbit

$$P_{k+1} = P_k + h\,r P_k \left(1 - \frac{P_k}{K}\right)$$

`P_k+1=P_k+h rP_k(1-_kK)`

Logistic model, Euler step

Worked examples, step by step

Example 9.1 Simulate a 20 m/s projectile at 45° and compare with the exact range.

```
import math
v0, angle, g, dt = 20.0, math.radians(45), 9.81, 0.001
x, y = 0.0, 0.0
vx, vy = v0 * math.cos(angle), v0 * math.sin(angle)
while y >= 0:
    vy -= g * dt
    x += vx * dt
    y += vy * dt
print(round(x, 2), "m simulated;", round(v0**2 / g, 2), "m exact")
```

1. Step velocity, then position, until it lands.
2. Exact range = $v_0^2 \sin 90°/g$.

Output

```
40.77 m simulated; 40.77 m exact
```

Answer: Both give 40.77 m to two decimals with dt = 0.001 s.

Example 9.2 Simulate one orbit 400 km above Earth with 1-second steps.

```
import math
GM = 3.986e14
r0 = 6.771e6
x, y = r0, 0.0
vx, vy = 0.0, math.sqrt(GM / r0)
dt, t = 1.0, 0
period = 2 * math.pi * math.sqrt(r0**3 / GM)
while t < period:
    r3 = (x * x + y * y) ** 1.5
    vx -= GM * x / r3 * dt          # Euler-Cromer: update velocity first
```

```
    vy -= GM * y / r3 * dt
    x += vx * dt
    y += vy * dt
    t += dt
print(round(period / 60, 1), "min;", round(math.hypot(x, y) / 1000), "km from Earth's center
```

1. Start at r_0 with circular speed $\sqrt{(GM/r_0)}$.
2. Update velocity from gravity, then position.

Output

```
92.4 min; 6771 km from Earth's center
```

Answer: After one period (92.4 minutes) the spacecraft is still 6,771 km from Earth's center: the orbit stays circular.

Example 9.3 Step a logistic fish population ($r = 0.5$, $K = 1{,}000$, $P_0 = 100$) for four years.

```
r, K, P, h = 0.5, 1000, 100.0, 1.0
for year in range(1, 5):
    P += h * r * P * (1 - P / K)
    print(year, round(P, 1))
```

1. Each year adds $rP(1 - P/K)$.

Output

```
1 145.0
2 207.0
3 289.1
4 391.8
```

Answer: 145.0, 207.0, 289.1, 391.8, matching Calculus II Lab A (391.9 there, a rounding difference).

Practice problems

Work each on paper first, then check.

1. What happens to the Example 9.1 error if dt is 0.1 s?

   The landing is detected late, so the range error grows; try it and compare.

2. Why does plain Euler make an orbit spiral outward?

It adds a little energy every step; Euler-Cromer nearly conserves it.

3. Add drag $-k\,v\,v_{\square}$ to Example 9.1. What happens to the best angle?

   It drops below 45° (Lab D).

Common misconceptions

Myth: A simulation that runs is correct.

Actually: Validate against exact cases and check that halving dt barely changes the answer.

Myth: Smaller dt is always better.

Actually: It costs time and can add rounding error; choose dt by convergence tests.

Myth: Models predict the future exactly.

Actually: They are approximations whose assumptions must be stated.

Flashcards

Time step dt

Size of each update.

Euler method

x += v·dt.

Euler-Cromer

Velocity first, then position.

Validation

Check against known answers.

Convergence

Answer stops changing as dt shrinks.

State

Variables that describe the system now.

Self-check quiz

1. Euler-Cromer updates...

position first

velocity first, then position with the new velocity

only velocity

nothing

velocity first, then position with the new velocity. Stable for orbits.

2. To validate a simulation...

trust it

compare with an exact solution

make dt huge

remove physics

compare with an exact solution. Known cases.

3. Exact range at 20 m/s, 45°, g = 9.81:

20.4 m

40.8 m

81.5 m

9.8 m

40.8 m. 400/9.81.

4. Halving dt and seeing no change means...

a bug

the result has converged

the model is wrong

nothing

the result has converged. Convergence test.

Try this quiz again

How this helps save the planet

Population models like Example 9.3 are part of how fishery scientists assess stocks and set sustainable catch limits.

The same time-stepping idea, scaled up enormously, is how weather and climate models are built.

On Saving Planets: Calculus II, Lab A · Orbits guide

Sources: NOAA Fisheries: population assessments (https://www.fisheries.noaa.gov/topic/population-assessments) · MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/)

Free learning resources

- MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/): Simulation and modeling lectures.
- NumPy: the absolute basics for beginners (https://numpy.org/doc/stable/user/absolute_beginners.html): Vectorizing simulations.

☐ **I finished this unit**

Problem Sets

These practice problems live inside each unit. Work on paper first, then open the answer.

1. Getting Started: Variables, Types & Expressions
   4 problems
2. Control Flow: Conditions & Loops
   3 problems
3. Functions
   3 problems
4. Lists, Dictionaries & Strings
   3 problems
5. Files & CSV Data
   3 problems
6. NumPy & Matplotlib
   3 problems
7. Debugging & Testing
   3 problems
8. Algorithms & Complexity
   3 problems
9. Numerical Simulation
   3 problems

Coding labs

Four coding labs use real public data and simulation: the Mauna Loa CO_2 record, NASA's global temperature record, a USGS river gauge, and a projectile with air drag.

Each lab's starter code runs as shown; the printed results were produced by running it. Then download the full data set and extend the analysis.

Safety reference: The official Python tutorial (https://docs.python.org/3/tutorial/)

Lab A: The Mauna Loa CO_2 record (NOAA)

WEEK 6

Goal: Load NOAA's annual-mean CO_2 CSV, compute the total rise and the recent growth rate, and plot the curve.

$$\text{rate} = \text{slope of } \text{np.polyfit}(\text{year}, \text{ppm}, 1)$$

```
rate=slope of np.polyfit(year, ppm, 1)
```

Growth rate from a linear fit

Procedure

1. Run the starter code (a few real rows from NOAA's file).
2. Download the full CSV from NOAA (link below), skip the # lines, and load all years.
3. Plot ppm against year with labeled axes and the source.
4. Fit lines to 1960–1969 and 2016–2025. How has the rate changed?

```
import csv, io
import numpy as np
text = """year,mean,unc
1959,315.98,0.12
1960,316.91,0.12
2020,414.21,0.12
2021,416.41,0.12
2022,418.53,0.12
2023,421.08,0.12
2024,424.61,0.12
2025,427.35,0.12
"""
rows = list(csv.DictReader(io.StringIO(text)))
yr = np.array([int(r["year"]) for r in rows])
ppm = np.array([float(r["mean"]) for r in rows])
print(round(ppm[-1] - ppm[0], 2), "ppm since 1959")
recent = yr >= 2020
slope = np.polyfit(yr[recent], ppm[recent], 1)[0]
print(round(slope, 2), "ppm per year, 2020-2025")
```

Output

```
111.37 ppm since 1959
2.65 ppm per year, 2020-2025
```

Data table

Year	Annual mean CO_2 (ppm)
1959	315.98
1960	316.91
2020	414.21
2023	421.08
2024	424.61
2025	427.35

Real values from NOAA GML's Mauna Loa annual-mean file, retrieved Oct. 24, 2026.

Sample calculation

1. 427.35 − 315.98 = 111.37 ppm since 1959.
2. Best-fit rate 2020–2025 ≈ 2.65 ppm per year.

> **Safety**
>
> - None beyond normal computer use.

Home or virtual alternative

- Colab or JupyterLite in a browser.

NOAA GML: Mauna Loa CO_2 annual means (CSV) (https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_annmean_mlo.csv) · Google Colab (free notebooks) (https://colab.research.google.com/)

Planet link: This is the record behind the Keeling Curve, the longest continuous measurement of rising CO_2. NOAA GML: Trends in atmospheric CO_2, data (https://gml.noaa.gov/ccgg/trends/data.html)

☐ **I completed this lab (wet or virtual)**

Lab B: Global temperature from NASA GISTEMP

WEEK 8

Goal: Analyze NASA's global annual temperature anomalies: averages, the warmest year and the trend.

$$\Delta T \text{ per decade} = 10 \times \text{slope}$$

```
Δ T\ per decade=10×slope
```

Trend per decade

Procedure

1. Run the starter code with the 2016–2025 annual (J–D) anomalies.
2. Download the full GISTEMP global CSV; skip its first title line; handle *** missing values.
3. Plot 1880–2025 and compute the trend since 1975.
4. Compare a 10-year fit with a 50-year fit and explain why short trends are noisier.

```
import numpy as np
years = np.arange(2016, 2026)
anom = np.array([1.01, 0.91, 0.85, 0.98, 1.01, 0.85, 0.89, 1.17, 1.28, 1.18])
print(round(anom.mean(), 3))
print(int(years[anom.argmax()]), anom.max())
print(round(np.polyfit(years, anom, 1)[0] * 10, 2), "C per decade (10-year fit)")
```

Output

```
1.013
2024 1.28
0.32 C per decade (10-year fit)
```

Data table

Year	Anomaly (°C vs 1951–1980)
2016	1.01
2020	1.01
2023	1.17
2024	1.28
2025	1.18

Real values from NASA GISTEMP v4 (GLB.Ts+dSST, J–D column), retrieved Oct. 24, 2026. NASA updates the analysis, so later downloads can differ slightly.

Sample calculation

1. Mean 2016–2025 = 1.013 °C.
2. A ten-year fit gives about 0.32 °C per decade, but ten years is short and noisy; use longer periods for the long-term trend.

> **Safety**
>
> - None beyond normal computer use.

Home or virtual alternative

- Colab or JupyterLite.

NASA GISTEMP v4 global means (CSV) (https://data.giss.nasa.gov/gistemp/tabledata_v4/GLB.Ts+dSST.csv) · NASA GISS: GISTEMP v4 surface temperature analysis (https://data.giss.nasa.gov/gistemp/)

Planet link: Reproducing NASA's record yourself is the best answer to "how do we know?" NASA GISS: GISTEMP v4 surface temperature analysis (https://data.giss.nasa.gov/gistemp/)

☐ **I completed this lab (wet or virtual)**

Lab C: A river through a USGS gauge

WEEK 9

Goal: Analyze daily streamflow from the USGS Potomac River gauge at Little Falls: units, statistics, recession and volume.

$$V = \sum_{i} Q_i \times 86{,}400 \text{ s}$$

V=_iQ_i×86,400\ s

Volume from daily mean flows

Procedure

1. Run the starter code with ten real days (June 1–10, 2025).
2. Request a full year from USGS Water Services (format=rdb) for site 01646500 and parameter 00060, and parse the tab-separated file.
3. Plot daily flow, find the peak, and compute the annual volume in km³.
4. Write tests for your unit-conversion function.

```
import numpy as np
cfs = np.array([44700, 40200, 29100, 22700, 17300, 14100, 12600, 12200, 14300, 22600])
m3s = cfs * 0.0283168
print(round(m3s.mean(), 1), round(np.median(m3s), 1))
drops = np.diff(cfs)
print(int(drops.min()), "cfs biggest one-day drop")
print(round(m3s.sum() * 86400 / 1e6, 1), "million m3 in 10 days")
```

Output

```
650.7 564.9
-11100 cfs biggest one-day drop
562.2 million m3 in 10 days
```

Data table

Date (2025)	Daily mean flow (cfs)
June 1	44,700
June 3	29,100
June 5	17,300
June 8	12,200

Date (2025)	Daily mean flow (cfs)
June 10	22,600

Real approved daily values from USGS site 01646500, retrieved Oct. 24, 2026.

Sample calculation

1. Mean 650.7 m^3/s, median 564.9 m^3/s (the mean is pulled up by the June 1 peak).
2. The largest one-day drop was 11,100 cfs; about 562 million m^3 passed in ten days.

> **Safety**
>
> - None beyond normal computer use.

Home or virtual alternative

- Colab or JupyterLite.

USGS: Potomac River near Washington, DC (Little Falls), site 01646500 (https://waterdata.usgs.gov/monitoring-location/USGS-01646500/) · USGS Water Services (data API) (https://waterservices.usgs.gov/)

Planet link: River gauges like this one support drinking-water supply, flood warnings and ecosystem flows. USGS: Potomac River near Washington, DC (Little Falls), site 01646500 (https://waterdata.usgs.gov/monitoring-location/USGS-01646500/)

☐ **I completed this lab (wet or virtual)**

Lab D: Projectile with air drag

WEEK 14

Goal: Extend the projectile simulation with quadratic drag and find the best launch angle as drag increases.

$$\vec{a} = -g\hat{y} - k\,|\vec{v}|\,\vec{v}$$

`a=-g y-k | v| v`

Quadratic drag (k per meter)

Procedure

1. Run the starter code, which searches whole-degree angles from 20° to 60°.
2. Check that k = 0 gives 45° and the exact range.
3. Halve dt and confirm the results barely change.
4. Plot range against angle for each k.

```
import math
def fly(v0, deg, k=0.0, dt=0.001, g=9.81):
    vx, vy = v0 * math.cos(math.radians(deg)), v0 * math.sin(math.radians(deg))
    x = y = 0.0
    while y >= 0:
        v = math.hypot(vx, vy)
        vx -= k * v * vx * dt
        vy -= (g + k * v * vy) * dt
        x += vx * dt
        y += vy * dt
    return x
for k in (0.0, 0.01, 0.05):
    best = max(range(20, 61), key=lambda a: fly(20, a, k))
    print(k, best, round(fly(20, best, k), 1))
```

Output

```
0.0 45 40.8
0.01 43 31.4
0.05 39 17.8
```

Data table

Drag k (1/m)	Best angle	Range (m)
0	45°	40.8

Drag k (1/m)	Best angle	Range (m)
0.01	43°	31.4
0.05	39°	17.8

Computed by the starter code (v_0 = 20 m/s, dt = 0.001 s).

Sample calculation

1. With no drag the best angle is 45° and the range 40.8 m (exact 40.77 m).
2. Drag shortens the range and lowers the best angle.

> **Safety**
>
> - None beyond normal computer use.

Home or virtual alternative

- Colab or JupyterLite.

Google Colab (free notebooks) (https://colab.research.google.com/) · JupyterLite: Python in your browser (Pyodide) (https://jupyter.org/try-jupyter/lab/)

Planet link: Drag is the same physics that sets an aircraft's fuel burn (Intro to Aerospace Engineering).

☐ **I completed this lab (wet or virtual)**

Exam prep practice sets

Time yourself: about 10 minutes per problem. Then compare with the full solutions.

Midterm practice set (Units 1–5)

1. What does print(7 // 2, 7 % 2) print?

    1. Floor division and remainder.

    Answer: 3 1

2. Write a loop that sums 1 to 100.

    1. total = 0; for n in range(1, 101): total += n.

    Answer: 5050

3. Write stall_speed(W, rho, S, cl_max=1.6) and evaluate it at 10,791 N, 1.227 kg/m³, 16.2 m².

    1. (2W/(ρSC))**0.5.

    Answer: 26.0 m/s

4. Given co2 = [418.53, 421.08, 424.61, 427.35], compute the year-to-year changes.

    1. [b - a for a, b in zip(co2, co2[1:])].

    Answer: [2.55, 3.53, 2.74]

5. Why does float("") need try/except when reading data?

    1. It raises ValueError.

    Answer: To skip or log bad rows instead of crashing.

Final practice set (all units)

1. Convert 44,700 cfs to m³/s.

    1. × 0.0283168.

    Answer: ≈ 1,265.8 m³/s

2. How many binary-search steps for one million sorted items?

    1. $\lceil \log_2 10^6 \rceil$.

    Answer: 20

3. How many bisection steps take [0, 2] below 10^{-6}?

    1. $2/2^n < 10^{-6}$.

    Answer: 21

4. Exact range of a 20 m/s projectile at 45° (g = 9.81)?

    1. v_0^2/g.

    Answer: 40.77 m

5. Four Euler steps of P += 0.5P(1 − P/1000) from 100?

    1. 145.0, 207.0, 289.1, …

    Answer: 391.8

6. Write a test for f_to_c.

1. assert f_to_c(212) == 100, and others.

Answer: Known pairs: 32 → 0, 212 → 100, −40 → −40.

Formula sheet

$$7//2 = 3,\ 7\,\%\,2 = 1,\ 2^{10} = 1024$$

$$\bar{x} = \tfrac{1}{n}\sum x_i$$

$$\text{rate} = \Delta y / \Delta x$$

$$y \approx mx + b \ (\text{np.polyfit})$$

$$1\ \text{cfs} = 0.0283168\ \text{m}^3/\text{s}$$

$$\text{binary search} \approx \lceil \log_2 n \rceil$$

$$n_{bisect} = \lceil \log_2((b-a)/\text{tol}) \rceil$$

$$v \mathrel{+}= a\,\Delta t,\ x \mathrel{+}= v\,\Delta t$$

$$R = v_0^2 \sin 2\theta / g$$

$$P \mathrel{+}= hrP(1 - P/K)$$

Constants

Constant	Value
1 cubic foot per second (cfs)	0.0283168 m³/s
Seconds per day	86,400
Mauna Loa CO_2, 1959 annual mean	315.98 ppm
Mauna Loa CO_2, 2025 annual mean	427.35 ppm
GISTEMP 2024 anomaly (vs 1951–1980)	1.28 °C
Earth's GM	3.986 × 10¹⁴ m³/s²
g	9.81 m/s²

Source: NOAA GML: Trends in atmospheric CO_2, data (https://gml.noaa.gov/ccgg/trends/data.html)

Free tools and references

Run Python free in your browser and look things up here.

- Google Colab (free notebooks) (https://colab.research.google.com/): Free notebooks with NumPy and Matplotlib preinstalled.

- JupyterLite: Python in your browser (Pyodide) (https://jupyter.org/try-jupyter/lab/): Python in the browser, no account, powered by Pyodide.
- The official Python tutorial (https://docs.python.org/3/tutorial/): The reference tutorial.
- MIT OpenCourseWare 6.100L: Introduction to CS and Programming Using Python (https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/): MIT's full intro course.
- Harvard CS50's Introduction to Programming with Python (https://cs50.harvard.edu/python/): Harvard's free Python course.

Glossary

Algorithm
Precise step-by-step method.

Array
NumPy container for fast numeric math.

Assertion
Check that stops the program if false.

Big-O
How cost grows with input size.

Bisection
Root finding by halving an interval.

CSV
Comma-separated values file.

Dictionary
Key → value mapping.

Edge case
Unusual input at a boundary.

Euler-Cromer
Stable time-stepping: velocity first.

Exception
Error raised during execution.

Function
Named, reusable block of code.

List
Ordered, mutable sequence.

Loop
Repeated execution.

Notebook
Document of runnable code cells.

Pyodide

Python compiled to run in a web browser.

Simulation
Stepping a model forward in time.

Traceback
Report of where an error happened.

Unit test
Small check of one function.

Variable
Name bound to a value.

f-string
Formatted string literal.

Self-assessment rubric

No grade or credit is given. Use this to judge honestly where you are in each unit.

Level	What it looks like
4 · Mastery	Writes clear, tested functions; validates results against known answers; explains choices.
3 · Proficient	Programs work with occasional bugs found by testing.
2 · Developing	Can modify examples but struggles to write from scratch.
1 · Beginning	Copies code without understanding; revisit the unit.

Suggested weighting if you want a score

Part	Weight
Practice problems	20%
Coding labs	35%
Midterm practice set	20%
Final practice set	25%

Aim for level 3 or better in every unit before moving to the next course.

Related courses

- Engineering Graphics / CAD ENGR 120
- Calculus III MATH 210
- General Chemistry I + Lab CHEM 110 + CHEM 110L
- All courses