CSC 1008

*CSC148!!

Week 1

Sadia ‘Rain’ Sharmin

Classes begin 10 minutes after the hour

Week 1

Welcome to CSC148

TODAY'S TOPICS:

- About the Course
- Memory Model
- Testing

Announcement

Before we get started...

We need a volunteer note-taker!

To become a volunteer notetaker, all you must do is attend classes regularly & submit/upload your notes consistently/weekly.

Volunteers can receive co-curricular credit(s) and/or a certificate of appreciation and/or be eligible for year-end prize draws.

Register Online as a Volunteer Note-Taker at:
https://clockwork.studentlife.utoronto.ca/custom/misc/home.aspx

More information on Quercus announcement.

Who am I?

A little about me

I've been teaching at UofT for 6 years (although this is my first year teaching at this campus).

This is my 6th time teaching CSC148.

I did my PhD on CS Education (to explore ways to teach you better!). I am interested in research involving mental wellness, inclusivity/diversity, and the pedagogy of kindness.

Also, I have 6 pets.

Sadia Sharmin
& Dr. Chirly the Parrotlet

let's get to know each other

2-minute "speed-dating" sessions

About the Course

About the course

What is CSC148?

By the end of this course, you will be able to:

- read and understand problem specifications,
- design and implement solutions to those problems,
- and evaluate your solutions for correctness, clarity, and efficiency,

and do all of these things like a computer scientist.

About the course

Things we will learn

Object-oriented design

Thinking recursively

Data structures

Program efficiency

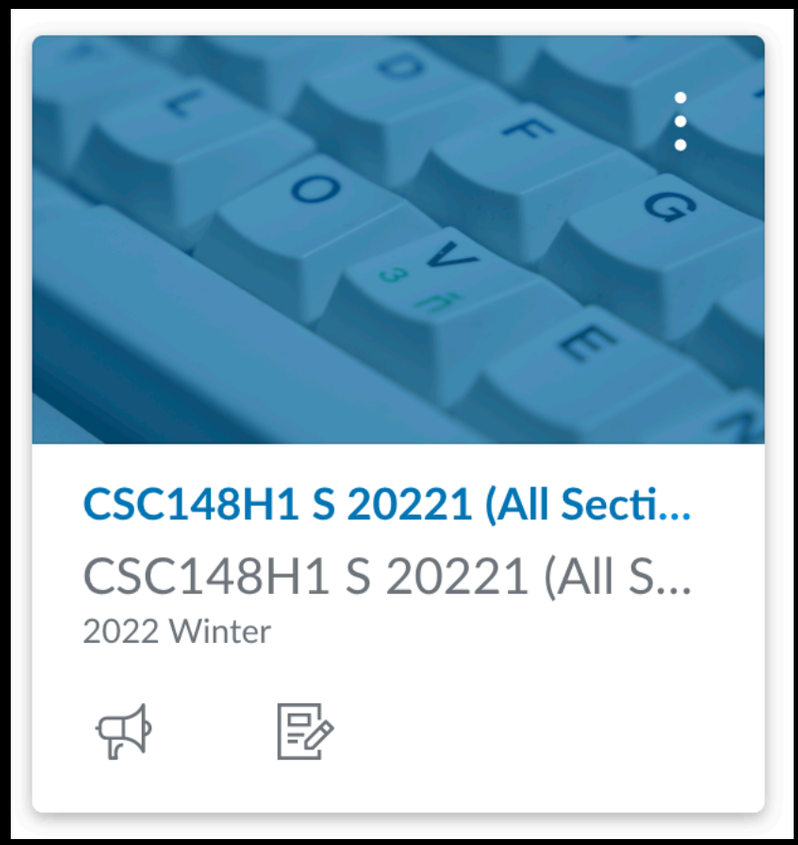
Theme: Abstraction

Theme: Professional practices

About the course

Where to find stuff

CSC148 is on Quercus



FAQ and link to Quercus at

https://www.teach.cs.toronto.edu/~csc148h/winter/

About the course

Preparation for CSC148

A recent credit in CSC108 is ideal prep for 148, but substantial programming experience (e.g., high school classes or work experience) is often an adequate substitute.

Check the course syllabus for links to resources and advice!

A ramp-up session this weekend can help you with Python, if you have experience in another programming language.

About the course

A Typical Week

CSC148 is semi-inverted.

Each week, you'll have:

- *Prep* readings and comprehension exercises
- *3 lecture hours* blending mini-lectures and active learning
- *2 lab hours* with larger programming tasks in a smaller group

About the course

Weekly Preps

Readings are where you'll initially see some of the content.

Accompanied by two kinds of comprehension exercises:

- Short-answer questions (Quercus)
- Programming tasks (MarkUs)

You'll succeed in 148 by:

- Completing preps before lecture

About the course

Lectures

Lectures are designed to promote active engagement with course content.

Active learning boosts exam grades [1, 2].

You'll succeed in 148 by:

- Actively working on problems in class
- Discussing with others
- Asking for help if you hit a snag

About the course

Labs

Labs are designed to provide larger technical programming tasks and practice quizzes under the guidance of a TA.

Practice quizzes boost learning skills and exam grades [3].

You'll succeed in 148 by:

- Maintaining your focus on the lab exercises
- Bouncing ideas around with others
- Asking questions of each other and your TA

Notes about lab logistics

You must sign up for a CSC148 TUT section on ACORN separately (***not*** the same as CSC148 LEC).

Labs begin Wednesday!

We'll post lab locations on Quercus.

About the course

Course Assessments

10 "prep" exercises	10%	Worth 1% each
9 labs (called "TUT" on Acorn)	9%	1% each. Synchronous and online until we can return to campus.
Research surveys	1%	Earn 1% by completing both surveys: one near the middle of term and one near the end.
3 Assignments	31%	A0 (5%), A1 (13%), A2 (13%)
Midterm	15%	Online during class time on Tuesday February 15th and Wednesday February 16th.
Final Exam	34% You must earn 40% or above on the final exam to pass the course; otherwise, your final course grade will be no higher than 47%.	During the final assessment period.

About the course

Assignments

Each CSC148 assignment is your opportunity to synthesize several course concepts into a fun and complex project.

A0 is solo.
For A1 and A2, you can work individually or with a partner.

Assignments are often the most challenging part of the course. Start early!!!

About the course

Academic Integrity

The work you submit must be your own.

Your work must not be submitted by anyone else.

Academic offences are taken very seriously.

About the course

Academic Integrity

Do discuss course concepts, what an assignment is asking, high-level ideas about the solution.

About the course

Academic Integrity

Don’t show anyone your code (including rough work).

Don’t copy code from *any* source.

About the course

Academic Integrity

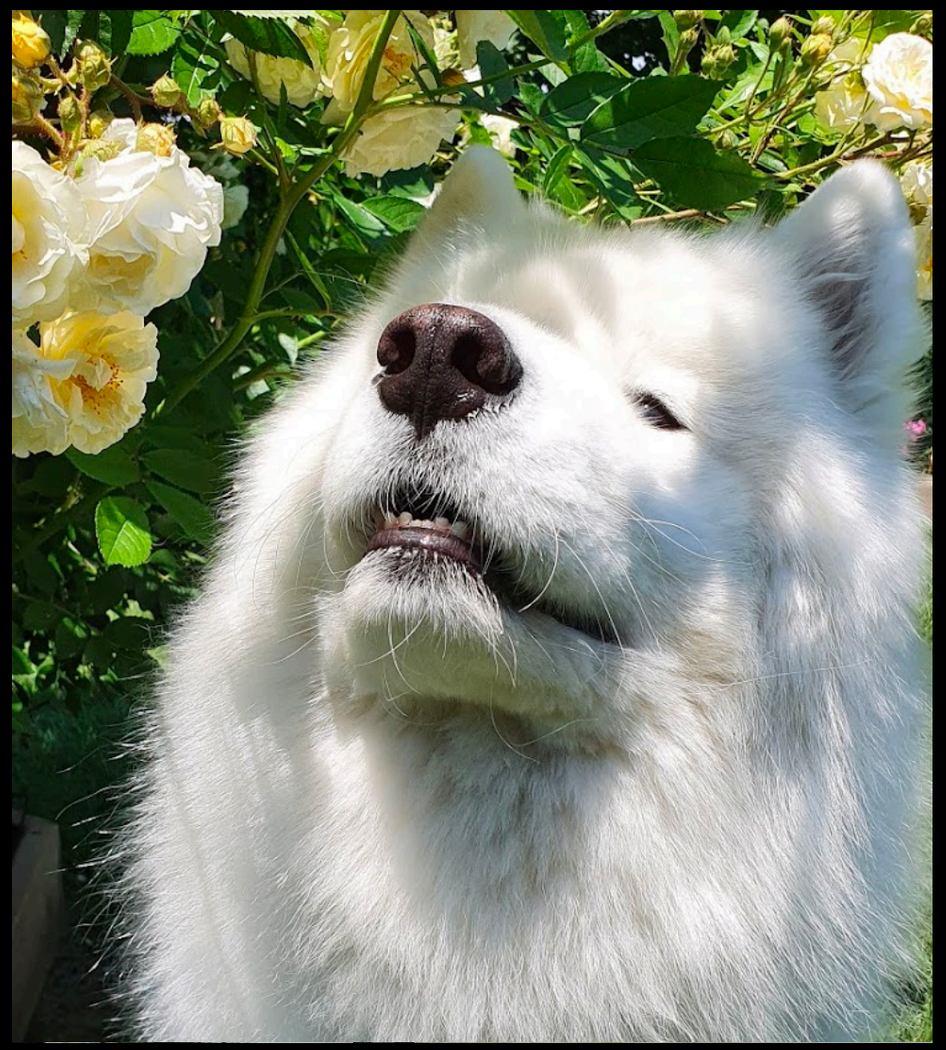
Do discuss course concepts, what an assignment is asking, high-level ideas about the solution.

About the course

Academic Integrity

Don’t show anyone your code (including rough work).

Don’t copy code from any source.



About the course

A word about the internet

It is fine to use the Internet to look up:

- Alternate presentation of course concepts
- Programming language documentation (docs.python.org)

But don't do any of the following:

- Copy code you find online
- Ask "How do I write this function…"
- Ask/pay someone to complete your work for you
- Show your code to a friend

About the course

Differences with CSC108

Preps

- Readings vs videos
- Culminating exercise is more challenging
- Submit on Markus vs PCRS; must test it yourself

There is no end-of-week “Perform”. We have labs instead.

Writing your own thorough tests is important to your success

About the course

Assignments vs other course work

Assignments are much larger, by design

- Class work, labs, and preps have limited scope
- Assignments are where you do something substantial
- If you have done the course work, you are ready

You'll succeed by

- Starting early
- Reading carefully
- Asking questions
- Working consistently
- Testing testing testing!

About the course

Getting help: don't be shy!

Instructor office hours

16 hours during the week

TA office hours

Schedule intensifies as due dates approach

"Group" office hours

For sharable questions about course concepts etc.

See Quercus for the full schedule, which varies by week!

Getting help: don't be shy!

Online course forum (Piazza)

- A good place to ask questions
- A good place to answer questions also!
- Monitored regularly by course staff

About the course

Stuck on software set-up?

There is a special office hour to help with this, this week:

- Thu 6-8
- Fri 4-6

See calendar here: https://q.utoronto.ca/courses/249810/external_tools/13713

We are also available to answer related questions on Piazza.

Tips for success

Prepare for, attend, and actively work in lectures and labs.

Start assignments early. Time-on-task isn't enough. You need *elapsed time* to:

- let ideas percolate
- get answers to questions that crop up

Don't spin your wheels. Come talk to us!

Practice, practice, practice. You are learning ways of thinking and new skills, and mastery of these will only come with lots of practice.

About the course

Active Learning

It is OK to not know how to solve a problem, or get an answer wrong

… but don't just wait for the answer without trying to figure out why you are stuck.

It is OK to need to take a short break, so you can refocus on class

… but do come back and refocus

It is OK to feel shy or nervous about working with new people

… keep in mind that most other people feel that way too!

About the course

Expectations

We expect that you will be respectful of your classmates' learning. This means:

- Participating in breakout discussions in a helpful and positive way
- Contributing on the chat without monopolizing it
- Minimizing non-class related discussion
- Contributing helpfully and respectfully on the discussion board

We expect that you will behave in a professional manner.

About the course

CSC148 ramp-up session

If you...

- took CSC108 some time ago and need a refresher,

  or

- have programming experience, but not in Python

then the CSC148 ramp-up session is for you!

- A free, 6-hour session
- Running twice: Saturday and Sunday 10-4pm
- Find out more, and register, on Quercus

About the course

Homework

Read our FAQ: https://www.teach.cs.toronto.edu/~csc148h/winter/

Review the course syllabus (on Quercus)

If appropriate, sign up for the Ramp-Up Session

Complete the CSC148 Software Setup (see the guide on Quercus)

If you haven't yet, complete Prep1! (unmarked, but required!!)

About the course

References

1. Active learning:
   Freeman et al, "Active learning increases student performance in science, engineering, and mathematics." PNAS 111 (23), 2014.

2. Diane Horton, Michelle Craig, Jennifer Campbell, Paul Gries, and Daniel Zingaro. 2014. "Comparing outcomes in inverted and traditional CS1". *ITiCSE 2014.*

3. Taking notes:
   Mueller et al, "The pen is mightier than the keyboard: Advantages of longhand over laptop note taking." Psychological Science, 2014.

4. Practice tests:
   Fernandez & Jamet, "Extending the testing effect to self-regulated learning." Metacognition Learning, 2016.

Memory Model

Recap

Python

Key terms

object

id/type/value

variable

refers to

assignment statement

alias

immutable

mutable

mutate

re-assign

WORKSHEET

Tracing simple code

PAGE 1

Memory Model

For Loop Gotcha

```
lst = [6, 2, 0, 44]

for item in lst:
    item = item + 1

print(lst)
```

What would be the output of this code?

A. [6, 2, 0, 44]

B. [7, 3, 1, 45]

C. None of the above

For Loop Gotcha

```
lst = [6, 2, 0, 44]

for item in lst:
    item = item + 1

print(lst)
```

Answer: A

Lesson: Do not assign to the loop variable in a for loop.

Memory Model

Functions

What happens when we call a function?

Memory Model

Functions

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Memory Model

Functions

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Memory Model

Functions

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Memory Model

Functions

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Memory Model

Functions

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Memory Model

New terms

frame

data about a function call (including local variables)

call stack

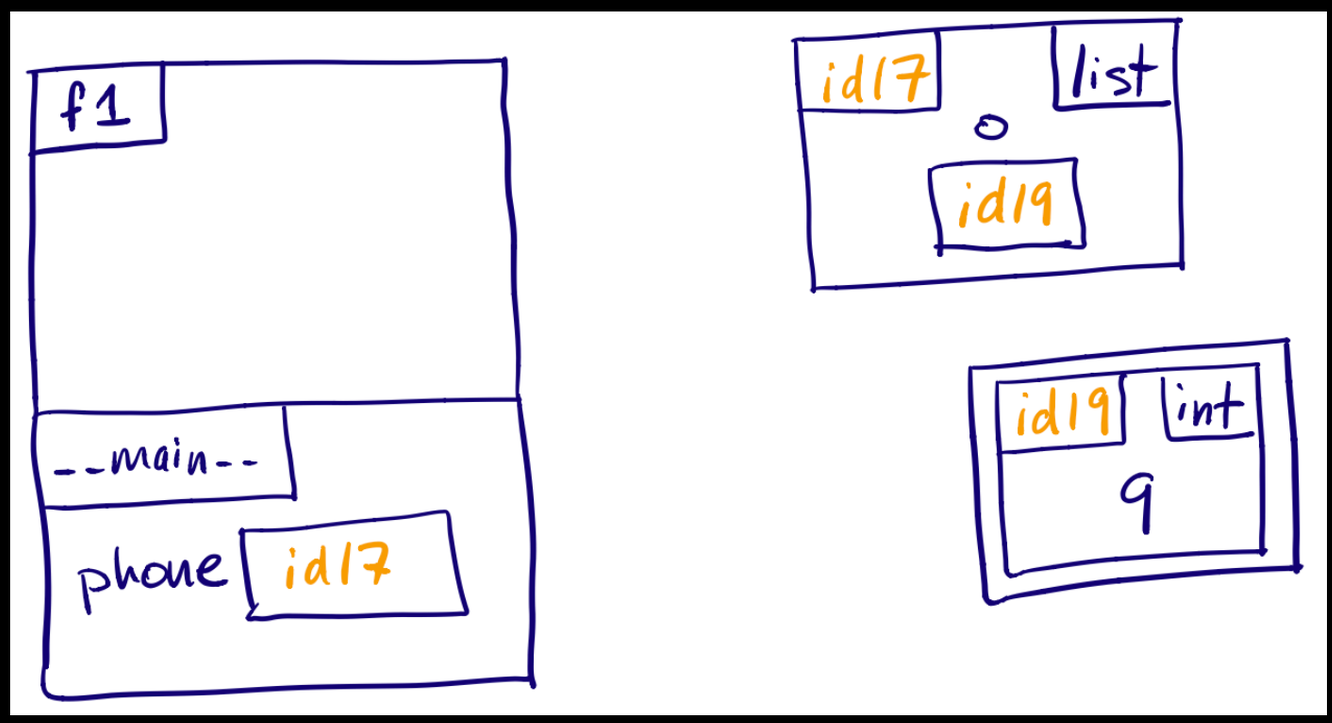
a collection of the frames of the functions that are *currently running*

Memory Model

Function Model

On the left, the call stack, whose frames show the program's variables and which ids they store

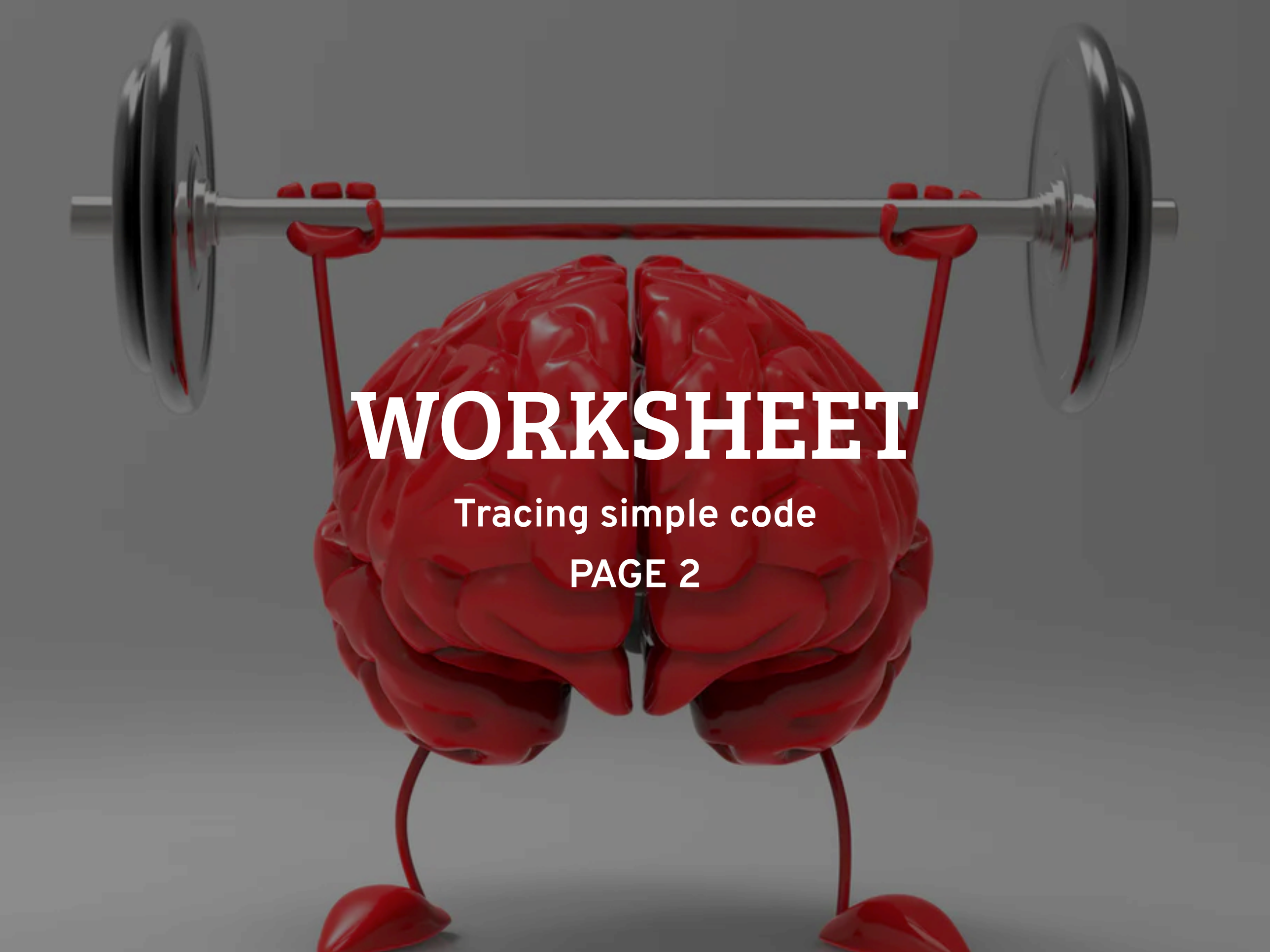
On the right, the object space, where all objects are shown (id, type, value)



Function Model

When a function is called, its parameters become *aliases* of the arguments passed to the function call.

If you understand this, you can do the rest of the worksheet!



WORKSHEET

Tracing simple code

PAGE 2

Memory Model

Function Gotcha

```
def f(x):
    x = 108

if __name__ == '__main__':
    course = 148
    f(course)
    print(course)
```

What would be the output of this code?

A. 108

B. 148

C. None of the above

Function Gotcha

```
def f(x):
    x = 108

if __name__ == '__main__':
    course = 148
    f(course)
    print(course)
```

Answer: B

Lesson: Do not assign to a function parameter in hopes of changing the argument.

Memory Model

Homework!

Predict the output of this code.

Trace it, draw a memory model of it, and then run it to see if you're actually right.

```
def f1(lst: List) -> None:
    lst[1] = 99
    lst = [100, 100, 100]

if __name__ == '__main__':
    grades = [60, 70, 80]
    f1(grades)
    print(grades)
```

Testing

Testing

Doctests

Purpose: showing examples of what the function does, useful for programmer writing client code

Doctests are included as example function calls within a function's docstring.

```
def insert_after(lst: List[int], n1: int, n2: int) -> None:
    """After each occurrence of <n1> in <lst>, insert <n2>.


    >>> lst = [5, 1, 2, 1, 6]
    >>> insert_after(lst, 1, 99)
    >>> lst
    [5, 1, 99, 2, 1, 99, 6]
    """
```

DEMO

Running doctests

See insert.py

Testing

Unit tests

Purpose: assessing units of code

"unit" = one function, usually

Unit tests are typically written in a separate file, enabling us to write a comprehensive set of tests without impacting readability of the code itself.

The key technical tools are:

- the assertion (Python: assert)
- the test case (Python: a function named test_*)

Testing

pytest

A module for automating unit tests.

An example:

NOTE: YOUR FUNCTION MUST BEGIN WITH "test_"

```
def test_simple() -> None:
    input_list = [5, 1, 2, 1, 6]
    insert_after(input_list, 1, 99)
    expected = [5, 1, 99, 2, 1, 99, 6]
    assert input_list == expected
```

1. SETUP
2. CALL
3. ASSERT

DEMO

Running pytest

See test_insert_example.py

Testing

Choosing test cases

This is the hard part (and the interesting part).

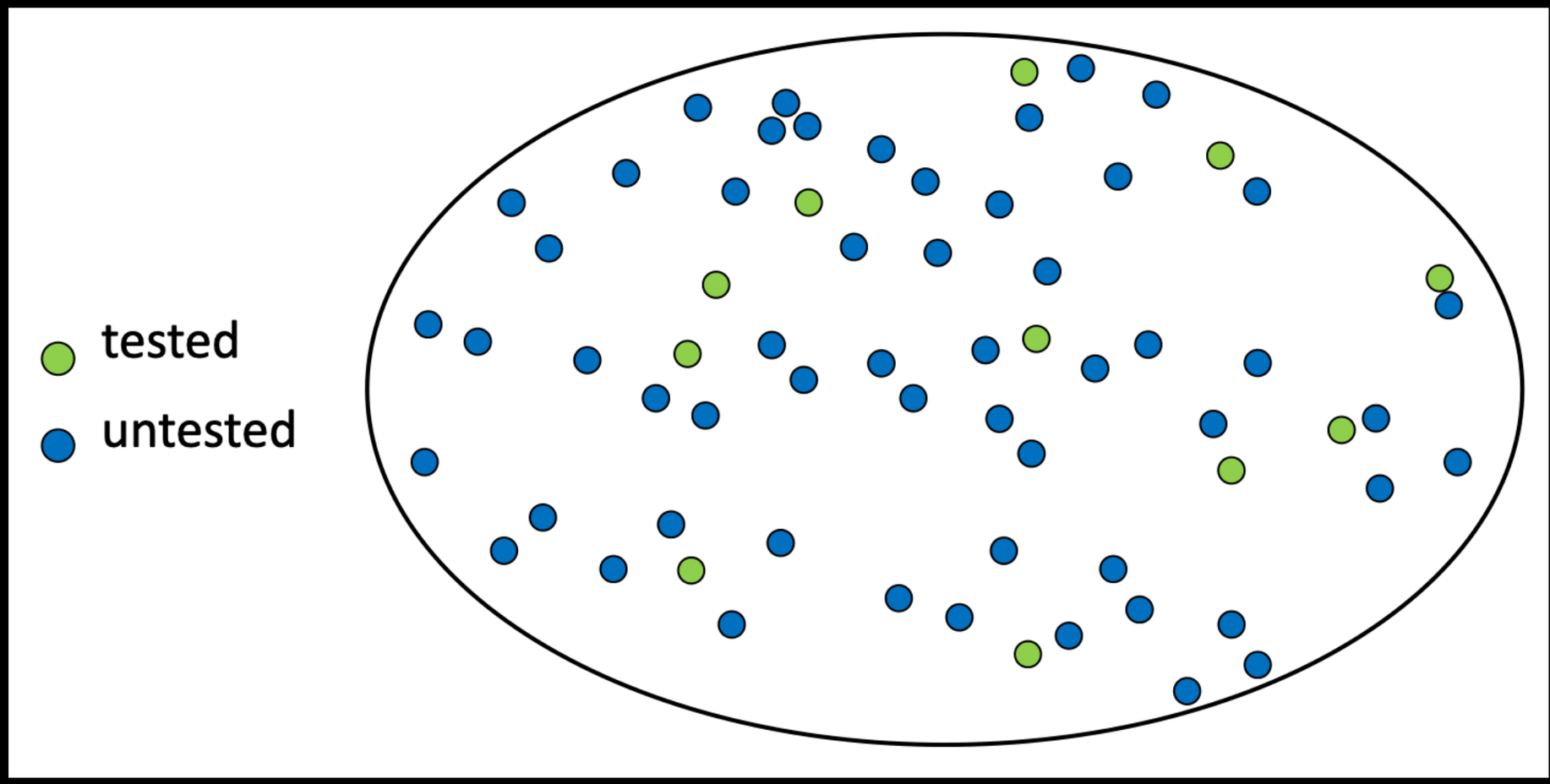
You are making an argument to *yourself* and any other reader.

You want to conclude that the code works on any valid input.

Testing

Choosing test cases

Choosing test cases randomly is futile.



Testing

Are we convinced?

We want to believe the function works on every possible input. But if the test cases are chosen at random, then we have no reason to believe the function works in the untested cases.

But we can't test each and every case!

Testing

Choosing cases

Are you confident the function works?

What if I said it passed 20 more test cases? 100 more cases?

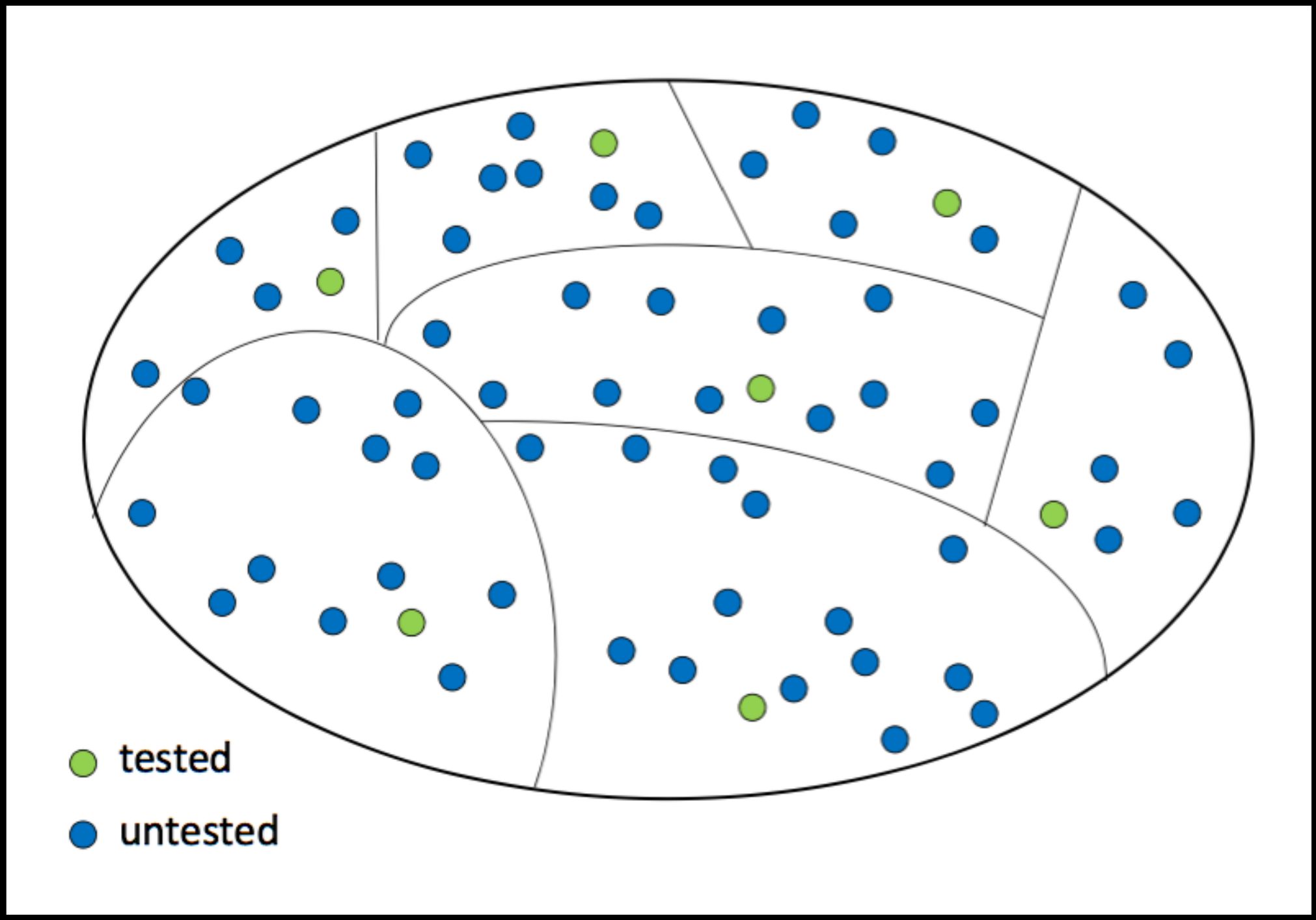
We generally cannot call a function with all possible parameter inputs e.g. if a function takes an integer as a parameter, there are an infinite number of values with which the function can be called

Luckily, quantity of test cases means little

Quality matters

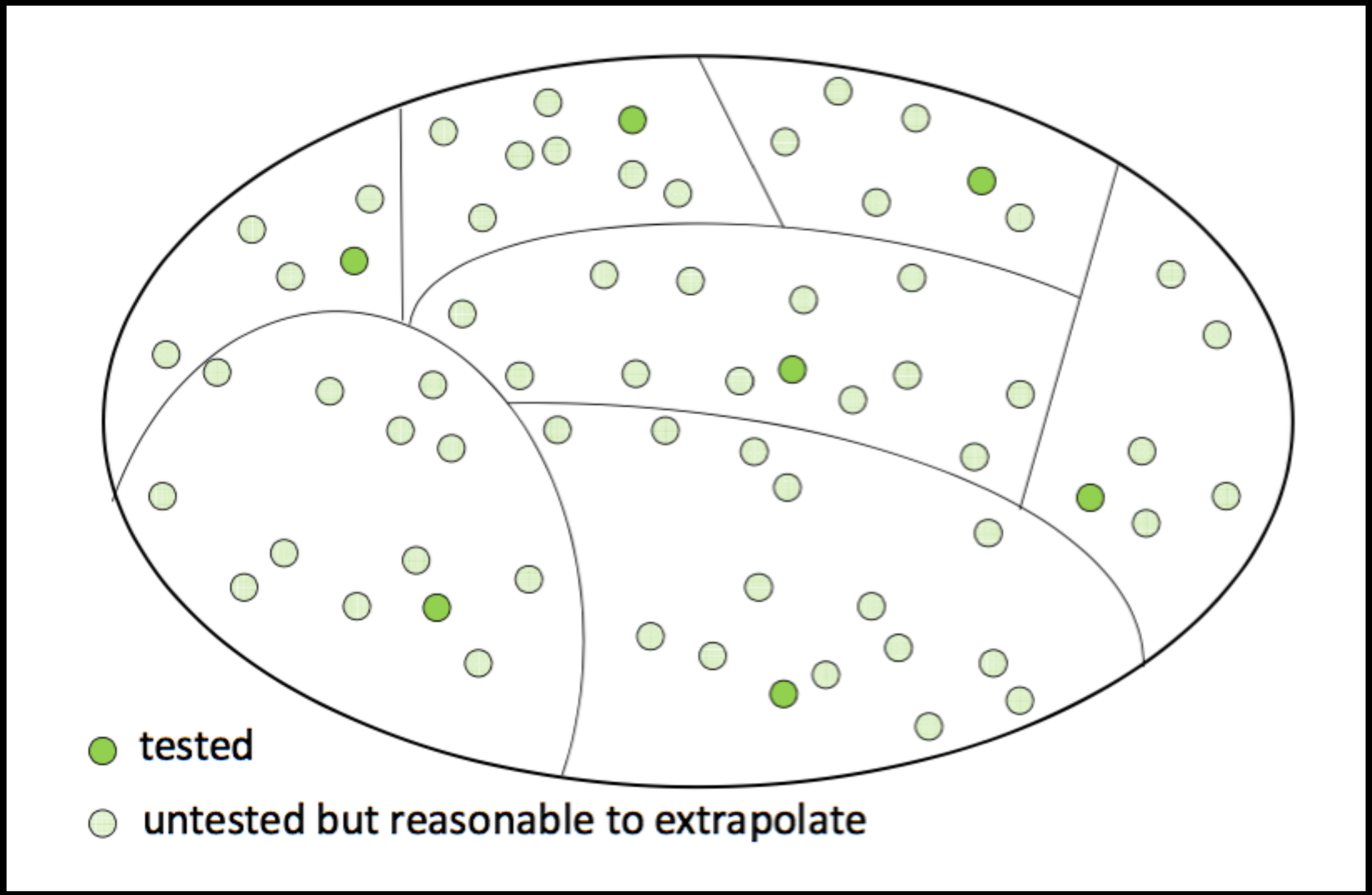
Testing domain

Carve the possible inputs into meaningful categories



Testing domain

Pick a representative from each category to test



Defining the categories

Test.ing

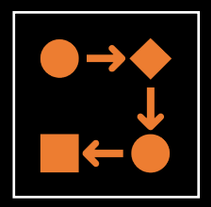
We base the representative categories on relevant properties of the input.

Things to consider:

Size

Boundary

Dichotomy



Order

Defining the categories

Testing

For collections (strings, lists, tuples, dictionaries) test with:

- empty collection
- a collection with 1 item
- smallest interesting case
- collection with several items

Size

Defining the categories

Testing

If a function behaves differently for a value near a particular threshold, test at the that threshold.

For example, if we were writing test cases for abs(num) which returns an absolute value of the given num, we would test at the boundary value -1, and those close to it like 0 and 1

Boundary

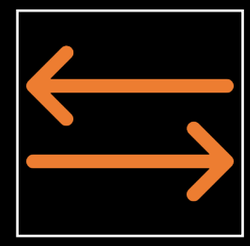
-1 is a boundary value because this is the input for which the function starts behaving differently

Defining the categories

Testing

Consider the different types of values that are relevant to the function, e.g.:

- vowels/non-vowels
- even/odd
- positive/negative
- empty/full
- etc.



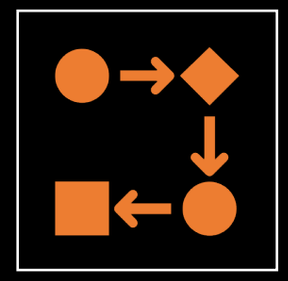
Dichotomy

Defining the categories

Testing

If a function behaves differently when the values are in a different order, identify and test each of those orders.

For example, unsorted order, non-decreasing, non-increasing, etc.



Order

Or, if we care about the relative location of two elements (e.g. adjacent, separated) or the position of something (beginning, end, elsewhere)

WORKSHEET

Choosing Test Cases

Testing

Property Tests

An alternate strategy rather than giving a specific input and expected output.

For example, suppose we are testing a function to find the max in a list:

	Instead of specifying this:	We specify this:
Input	[3, 62, 4, 53, 9]	A list of integers
Output	62	An element of the list, or None

Inputs (many!) are generated and tested automatically.

DEMO

Property-based testing with `hypothesis`

See insert_hypothesis.py