CSC 148

Introduction to Computer Science

Testing with purpose

How do we **identify** problems with our code?

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

A beginner's way to test a function

- Write calls in the console
- Read the results and judge whether correct
- What are the disadvantages of this?

Doctests: tests for user understanding

```
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]
    """
    ...

if __name__ == ‘__main__’:
    import doctest
    doctest.testmod()
```

Unit tests: tests for "units" of code

- With doctests, thorough testing would make docstrings too long
- You will have seen unit tests a bit in 108 ...
- "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 whose name begins with "`test_`")

Unit tests: tests for “units” of code

- Example:

```
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
```

- Further documentation:
  - https://docs.python.org/3/library/unittest.html

Pytest: simple and powerful test framework

- Simplifies writing small tests (not as much code to write as unittest), but still powerful for complex testing
- Expects tests to be in separate files that begin with test_ or end with _test.py:

cat.py

```python
def meow(n: int) -> str:
    say = 'meow'
    return n * say
```

test_cat.py

```python
from cat import meow
import pytest

def test_meow() -> None:
    assert meow(2) == 'meowmeow'

if __name__ == '__main__':
    pytest.main(['test_cat.py'])
```

- Not in the standard library
    - Sometimes that's a good thing :)
- Further documentation:
    - https://docs.pytest.org/en/latest/

Main goal of this lecture

- Lots of testing frameworks out there, learning a few is useful

- We expect you to know how to use doctest and pytest
  - Use documentation
  - Practice examples

- However, most important (and challenging) skill to learn is knowing how to choose (good!) test cases
  - We will focus on this next!

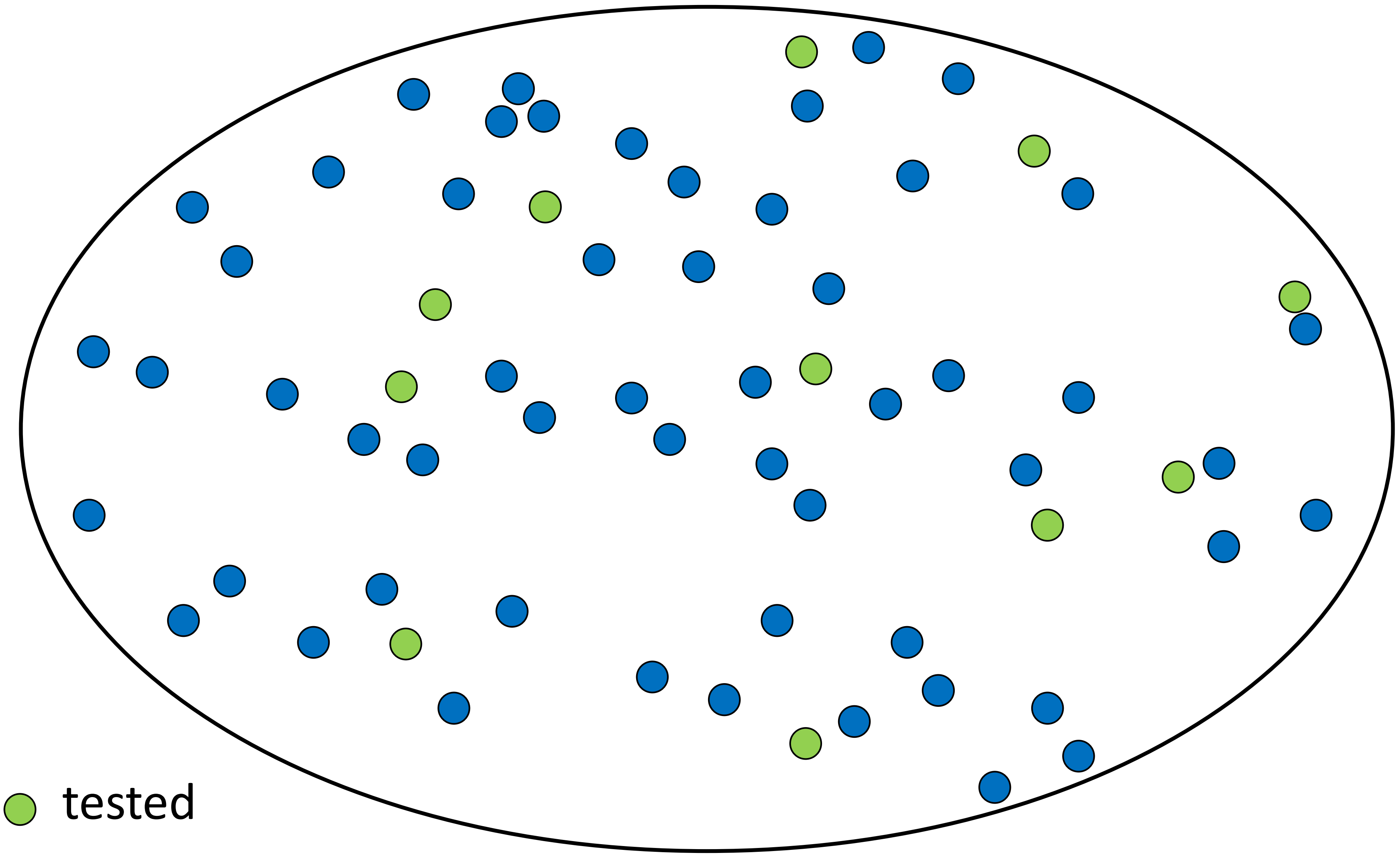
We will focus on **choosing** test cases

- Example: a function to find the maximum in a list
- Test cases:

List	Expected Result	Test passed?
[3, 6, 4, 42, 9]	42	yes
[22, 32, 59, 17, 18, 1]	59	yes
[1, 88, 17, 59, 33, 22]	88	yes
[1, 3, 5, 7, 9, 1, 3, 5, 7]	9	yes
[7, 5, 3, 1, 9, 7, 5, 3, 1]	9	yes
[561, 1024, 13, 79, 97, 4]	1024	yes
[9, 6, 7, 11, 5]	11	yes

- Are you confident the function works?

Testing domain

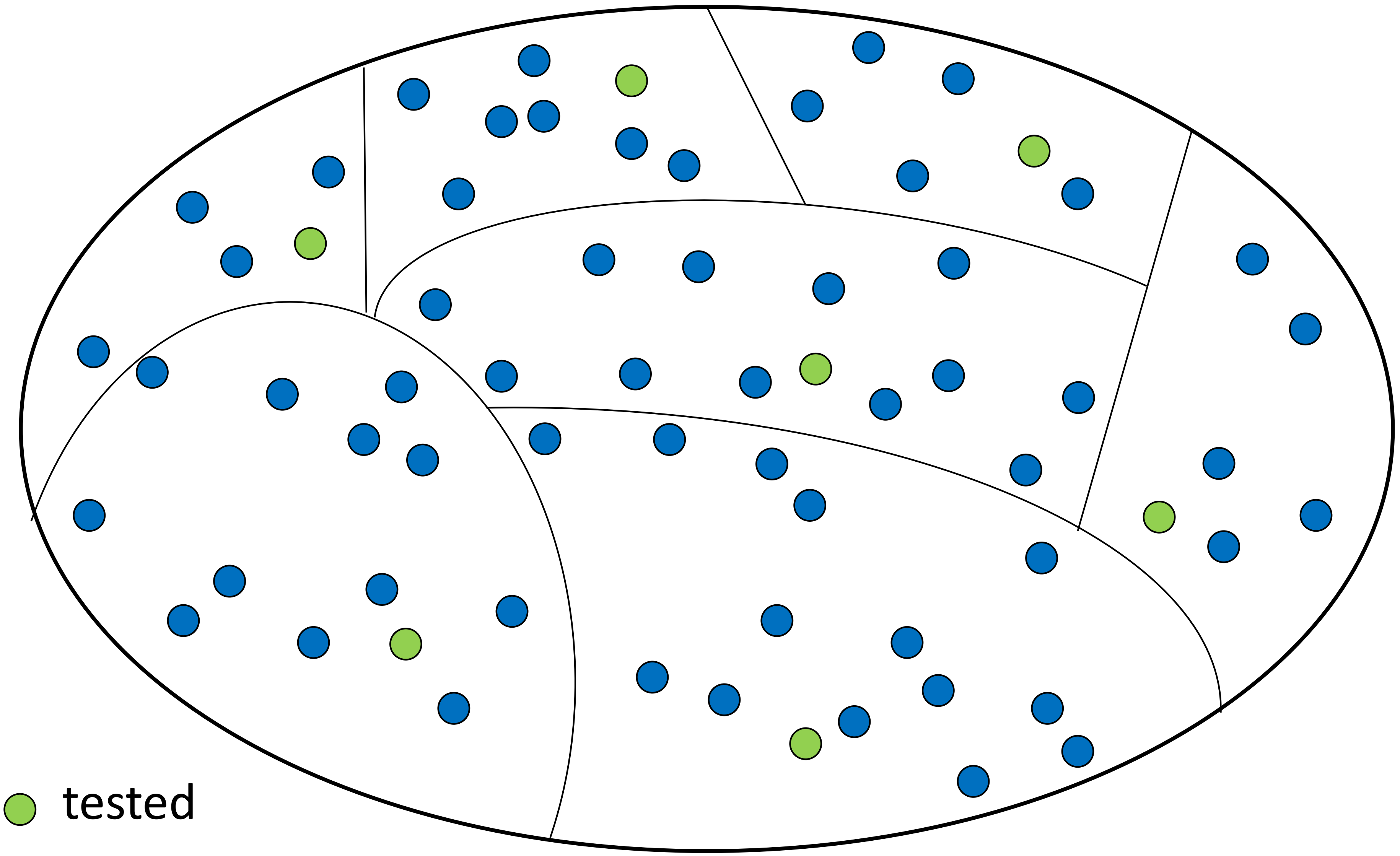


● tested

● untested

(In reality there are many more possible test cases)

Testing domain

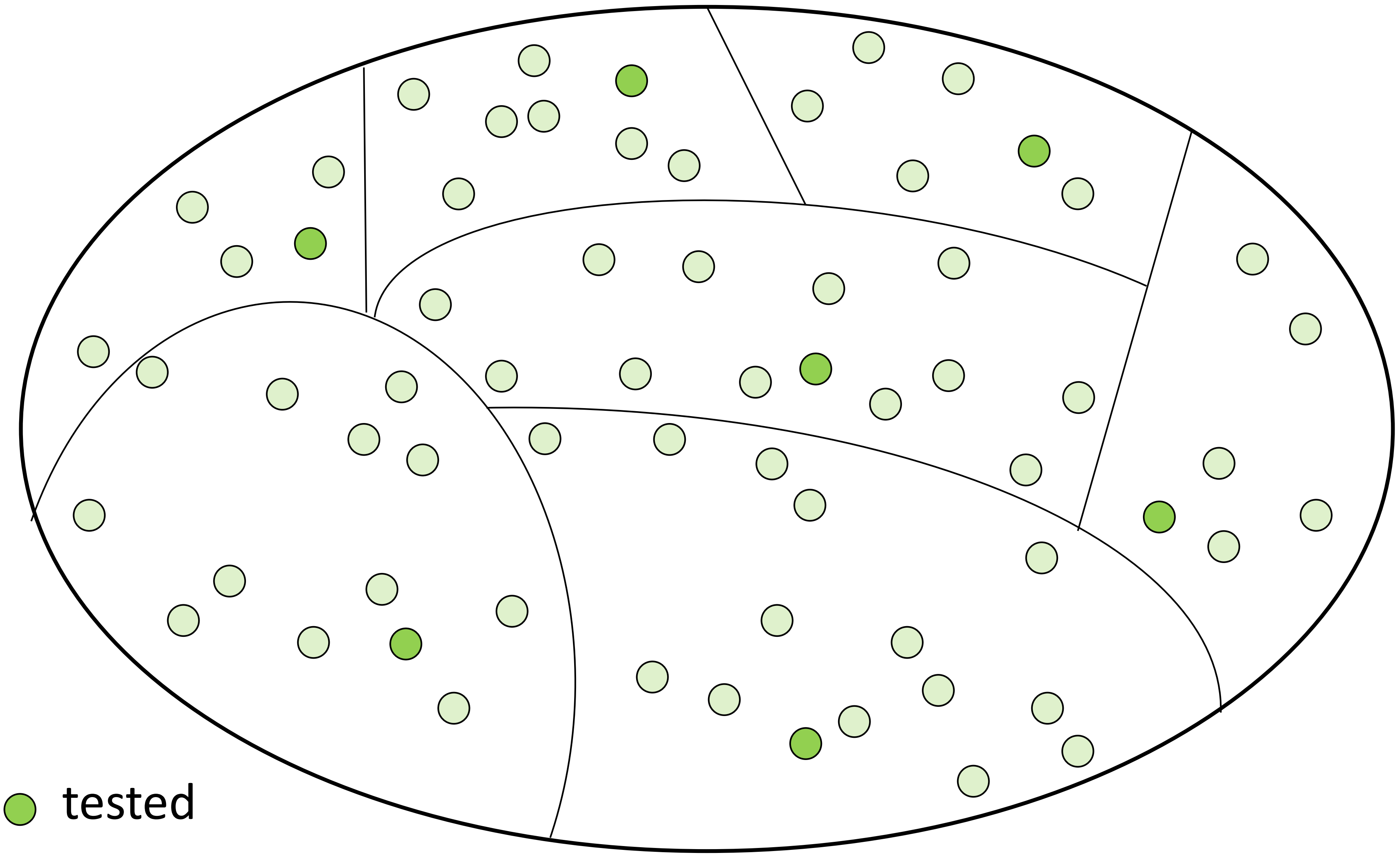


● tested

● untested

(In reality there are many more possible test cases)

Testing domain



● tested

○ untested but reasonable to extrapolate

But which properties?

- We have to decide on which properties are relevant – there may be so many!
- Decide based on knowing what a function or method does
- If we know how it does it, that can influence our choices too
  - e.g., if a method divides a list in half, odd vs even size is pretty important!

Choosing input properties

```python
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]
    """
```

- Input properties
- Example:
  - position of `n1` in `lst` (front, middle, back)
- Worksheet ...

Property tests: describing behaviour

- Generating random inputs is easy, but it's time-consuming to check correctness on each input directly
- So instead describe properties of the desired function behaviour, and check these properties on a huge number of random inputs

Instead of specifying this:	We specify this:
A specific input, e.g., [3, 6, 4, 42, 9]	A category of input, e.g., lists of integers
A specific output, e.g., 42	A property of the output, e.g., returns an element of the list or None

Thoughts on testing

- Designing test cases before writing code is a best practice in industry
- It is part of test-driven development
- When you test code, you must try to break it!

Fixing a bug

- When your testing reveals a bug, what to do?
- Beginners often:
  - Try some “typical” changes, e.g., change ">" to ">="
  - Add print statements
- A rarely done but better strategy:
  - Trace the code on paper
  - Why is this better?
- A professional strategy:
  - Use the debugger to trace it for you
  - Use what you learn to hypothesize a fix

Checking your fix

- Reap the benefit of having defined a thorough set of tests
  - Called a "test suite"

=> You can now check any new code change or fix with the press of a button!

Professionalism

- We have seen two practices that are expected of any professional:
  - Test-driven development
  - Using a debugger to find and fix bugs
- You will hone these skills throughout the course

- *Professionalism is a theme we will revisit*

Your first lab was Thursday … or today!

- Reminders:
  - You must go to the timeslot you signed up for on Acorn!
- Review lab policies on the course syllabus.
- Labs are graded based on participation
  - Show up on time!
  - Work hard on the lab exercises!
  - Complete the quiz and discussions in the second part of the lab!
- For each lab, you must work with a partner/group (unless you have an accommodation)
  - Can be different classmates each time if you wish

Announcements / reminders

- Post-lecture tips: revisit this week's readings to "solidify" what you've learned!
  - Readings 1.1-1.7
- Prep 2 is out (and for credit!)
- Lab 1: software installation can be annoying, but better do it now, than when you need to actually start working on an exercise or an assignment!
  - You don't get extra time on preps/labs/assignments due to "technical difficulties"!