

LEAP Life - Computer Literacy – January 5, 2026

Day 1



Department of Electrical & Computer Engineering



Ed Solovey – Associate Professor of the Practice - esolovey@bu.edu

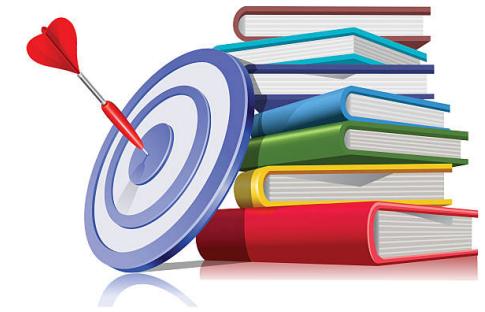
- MIT '01 – Computer Science, Bachelor's of Engineering
- MIT '03 – Computer Science, Master's of Engineering – Theory of Computation
- Twenty+ Years of Software Experience



- Oracle
- Adobe
- Brightcove
- Crashlytics
- Twitter
- Google
- Digits

Program Overview

- Day 1
 - Python Overview
 - GitHub Set Up
 - Effective LLM Collaboration
 - Code Comprehension & Organization
 - Testing & Continuous Integration
- Day 2
 - Software Development Processes
 - Code Runtime
 - Cloud Computing & Continuous Deployment
 - Web Servers and API's
- Day 3
 - Final Projects - Continuous Deploying a Web Server / API to Amazon Web Services



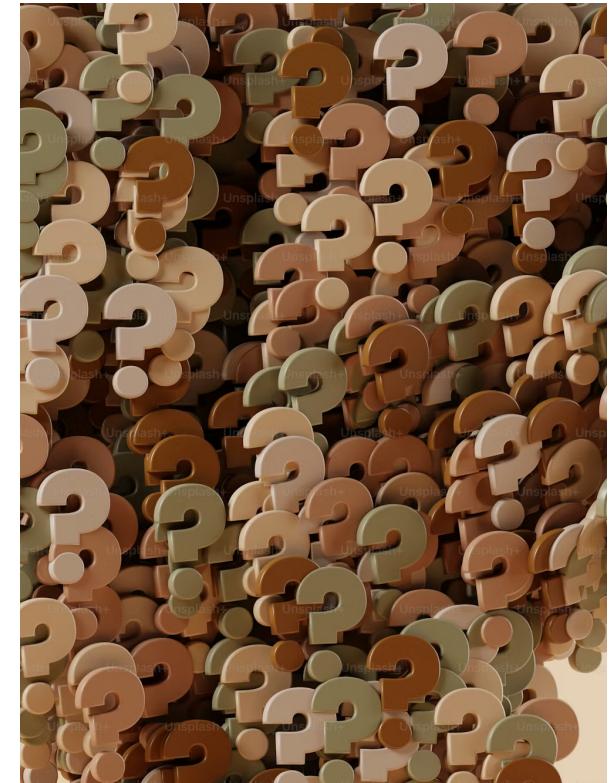
Philosophy & Disclaimer

- We will go over concepts faster than we can explain them all with a ton of detail, but will circle back to them, and ...



Philosophy & Disclaimer

- Please ask questions!!



Philosophy & Disclaimer

- Let's be super collaborative!

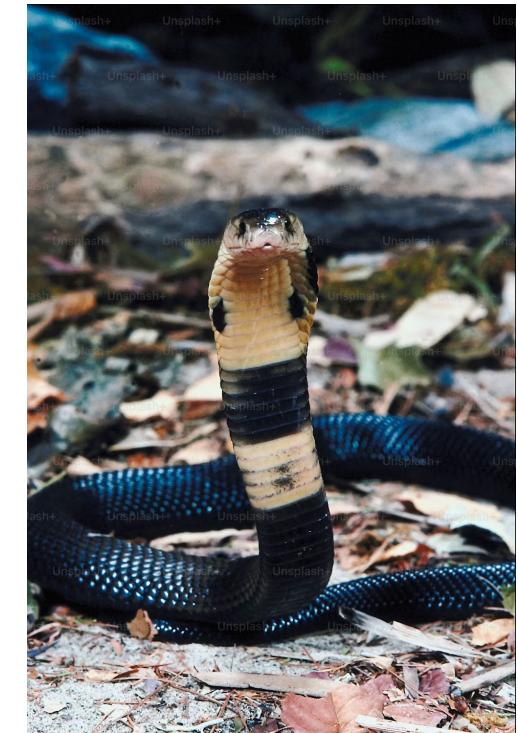


Introductions!



Day 1 - Agenda

- GitHub
 - Forking **base** repo and setting up your own repo
- Dissecting a program
- Installing PyCharm
- Introduction to testing
- Re-factoring to improve testability
- Continuous Integration (via GitHub Actions)
- Re-factoring to improve readability, code organization
- Object Oriented (Basics) to improve code organization



Effective LLM Collaboration



S staff main Trial

local/main.py

```

1 from common.exam_data_manager import read_and_compute
2 from common.exam_data_processor import ExamDataProcessor
3 from local.local_csv_data_fetcher import LocalCSVDataFetcher
4
5 import os
6 import json
7
8 INPUT_FILENAME = "local/test_scores.csv"
9 OUTPUT_FILENAME = "local/output.json"
10
11 if os.path.exists(OUTPUT_FILENAME):
12     os.remove(OUTPUT_FILENAME)
13
14 result = read_and_compute(
15     LocalCSVDataFetcher(INPUT_FILENAME),
16     ExamDataProcessor()
17 )
18
19 with open(OUTPUT_FILENAME, "w") as out:
20     json.dump(result.to_dictionary(), out, indent=2)
21

```

BU

1:18 LF UTF-8 4 spaces Python 3.11 (staff)

ChatGPT 5.2 > How do I apply a function to every element of a Python list?

In Python, there are three canonical ways to apply a function to every element of a list. Which one you choose depends on whether you value clarity, functional style, or iterator efficiency.

1. List comprehension (recommended, most Pythonic)

This is the clearest and most commonly used approach.

```
python
def square(x):
    return x * x

nums = [1, 2, 3, 4]
squared = [square(x) for x in nums]
```

Why this is preferred

- Very readable
- Explicit about creating a new list
- Easy to add conditions later

```
python
squared_evens = [square(x) for x in nums if x % 2 == 0]
```

2. map() (functional style)

```
python
squared = list(map(square, nums))
```

or with a lambda:

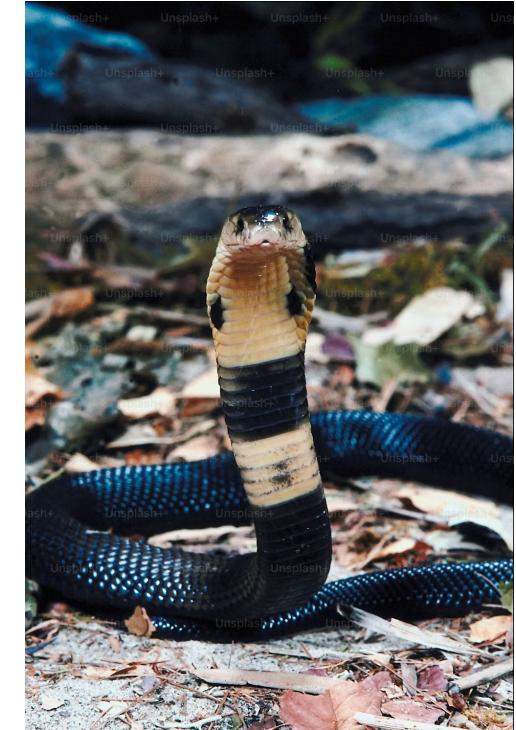
Ask anything



Day 1 - Agenda

- GitHub

- Forking base repo and setting up your own repo



What can go wrong while managing code for a one-person team?

- Implemented a new feature. Saved changes to a bunch of files, but discovered a problem.
Want to start over, but how?
- Tried to re-factor some messy code, but plan did not work out. How do I get back to what I had?
- Accidentally deleted a file.
- Computer/Hard-drive crashed/lost.



What can go wrong while managing code for a one-person team?

- Can try to manually:
 - make copies of all files before big changes
 - have a file naming schema, e.g. MyClass-Jun12.java etc...
 - back-up frequently to cloud or external drives

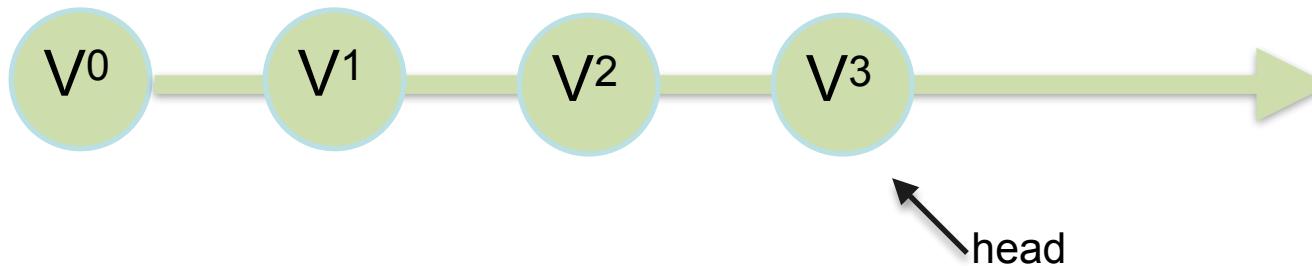


What can go wrong while managing code for a one-person team?

- Manual approach would
 - be overwhelming
 - time consuming - you should be spending time writing code not managing copies of it
 - would eventually fall over completely



Source Code Specific Requirements



- Each version (V) in the image above represents a version/snapshot of your entire codebase (repository)
- An individual file, e.g., `AnalyticsComputer.java` may have changed in each version, or just a subset of them
- **head** is a pointer to the current, best version of your repository

Source Control Evolution History : Locking Strategy



- Jill and Jane are both interested in making changes to a common utility file
- Do we
 - require that one of them reserves a Lock before making their changes OR
 - allow them to make changes in parallel and figure out how to combine them later?

Repos & Branches

server: GitHub

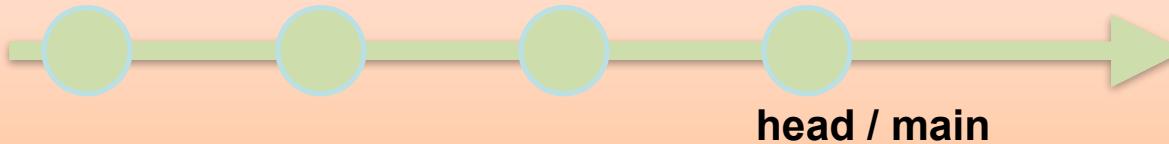


git clone (very first time)

git pull (subsequent times to get latest from server)

Repos & Branches

server: GitHub



developer machine

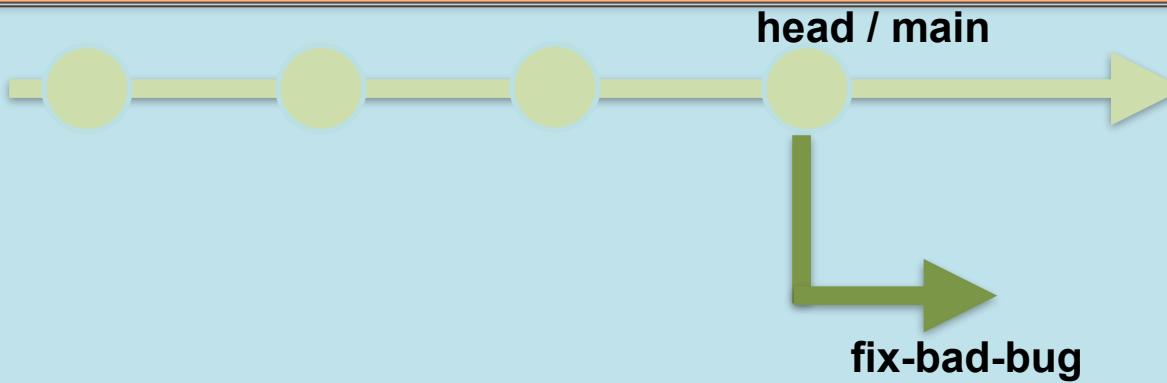


Repos & Branches

server: GitHub



developer machine



git branch fix-bad-bug // creates a new branch locally



ctrical & Computer Engineering

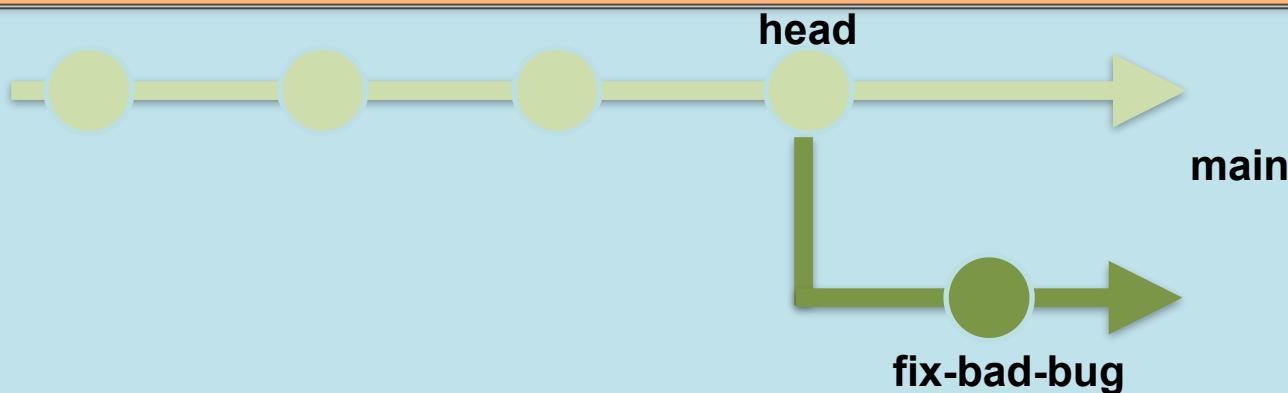


Repos & Branches

server: GitHub



developer machine



git commit -a -m “improved sorting algorithm”
// commits in-progress changes to local branch



ctrical & Computer Engineering

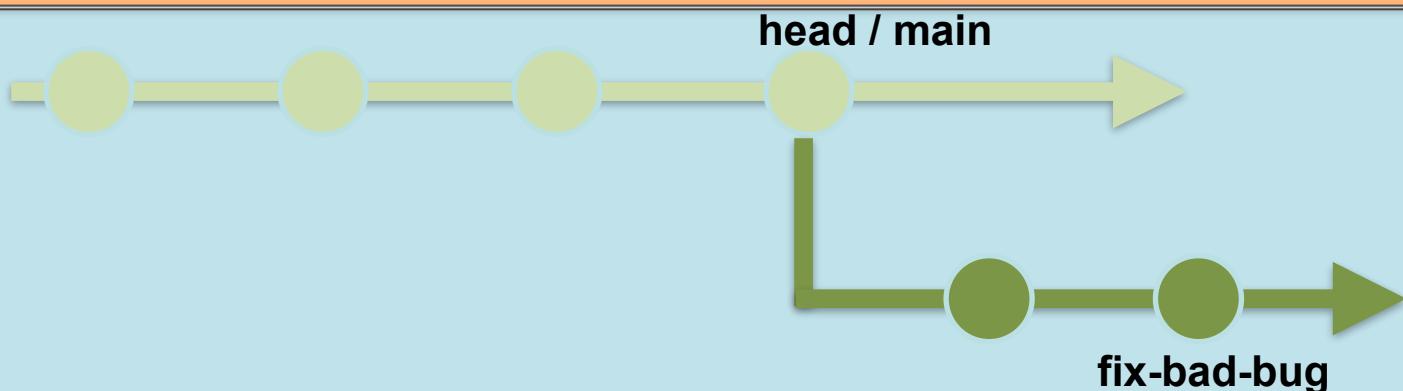


Repos & Branches

server: GitHub



developer machine



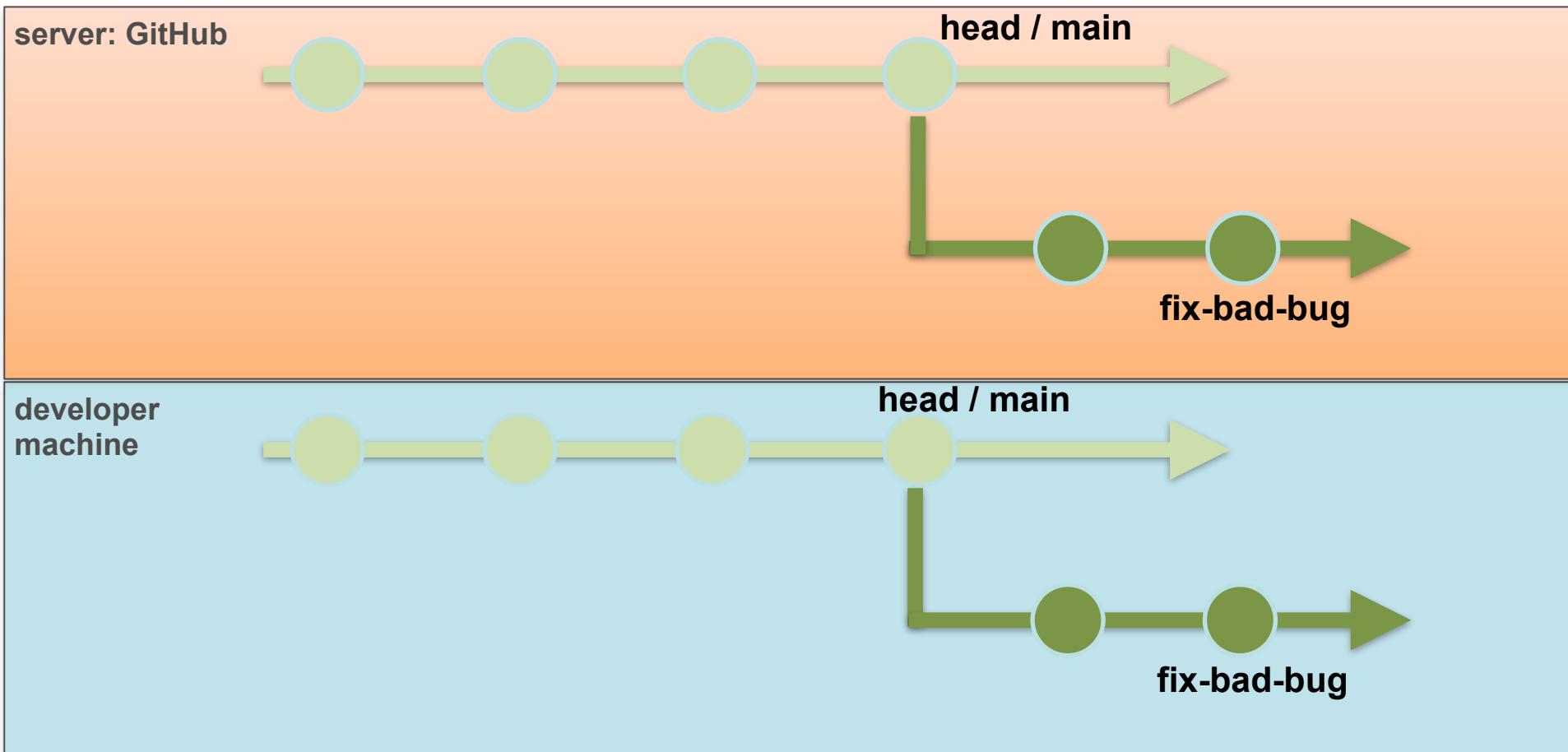
git commit -a -m “added alternate sorting algorithm”
// commits in-progress changes to local branch



ctrical & Computer Engineering



Repos & Branches



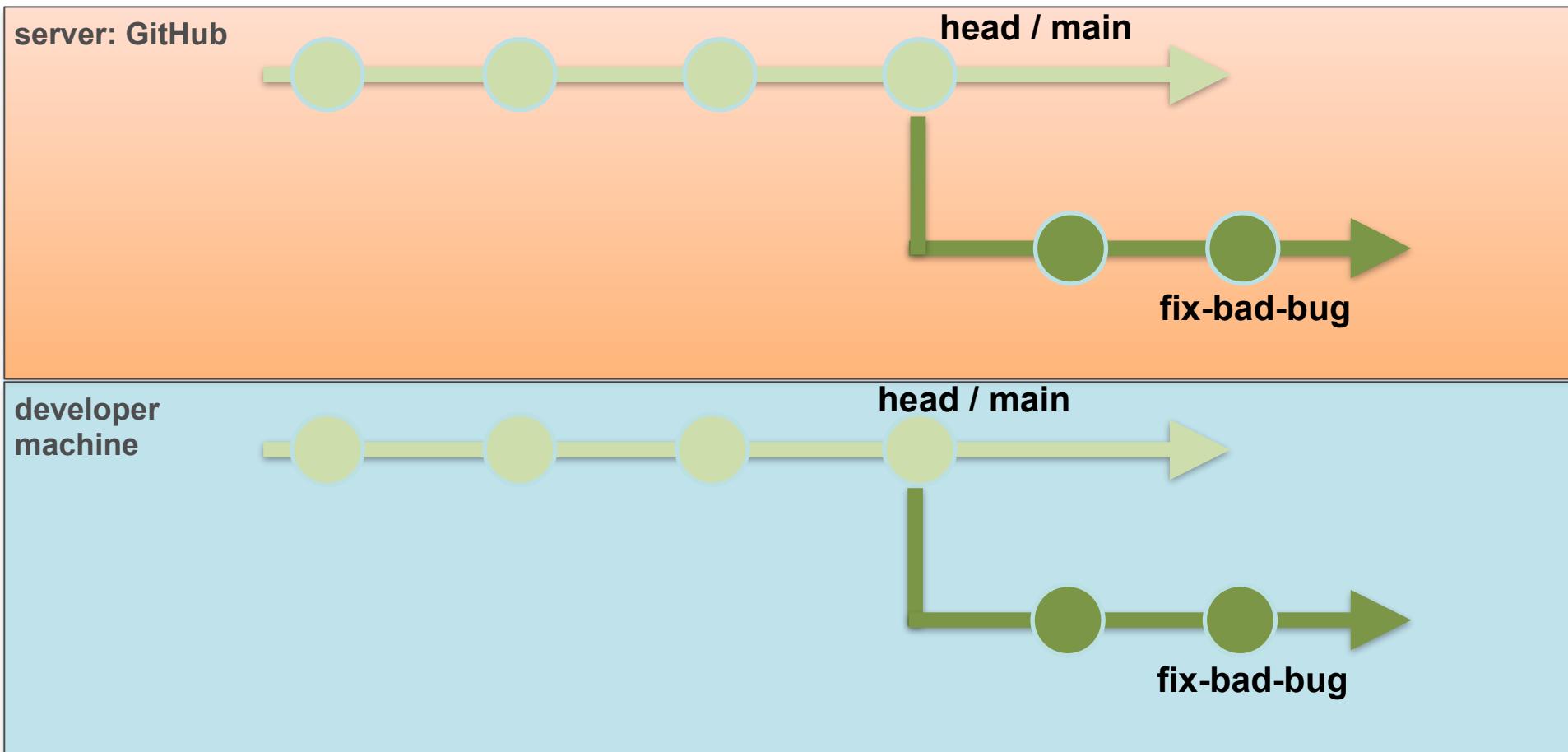
git push --set-upstream origin fix-bad-bug
// pushes new branch up to the server



ctrical & Computer Engineering



Repos & Branches



creates pull request to merge **fix-bad-bug** into **main**



ctrical & Computer Engineering

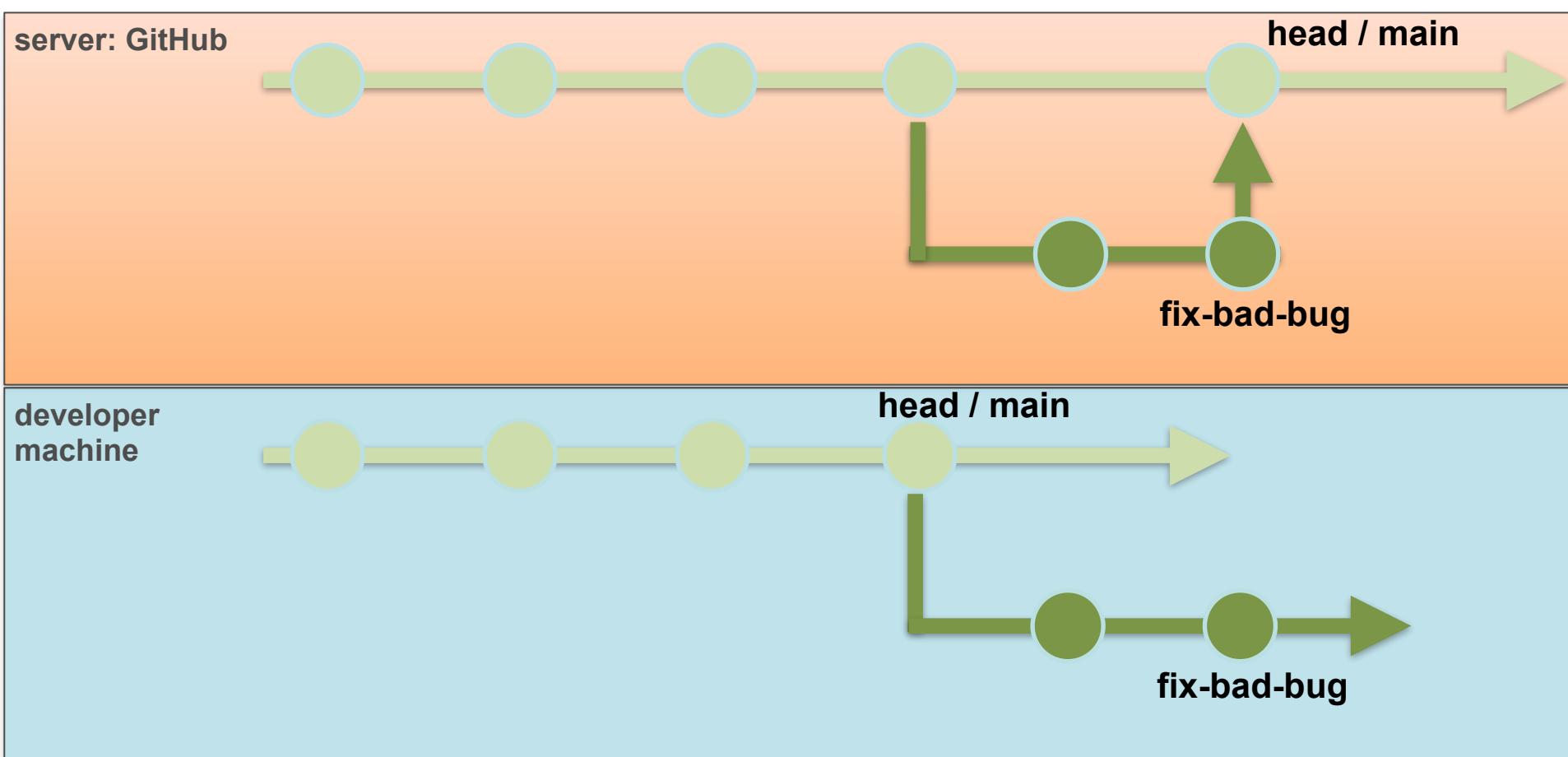
Open a pull request

Create a new pull request by comparing changes across

base: main ▾ ← compare: branch_example ▾ ...

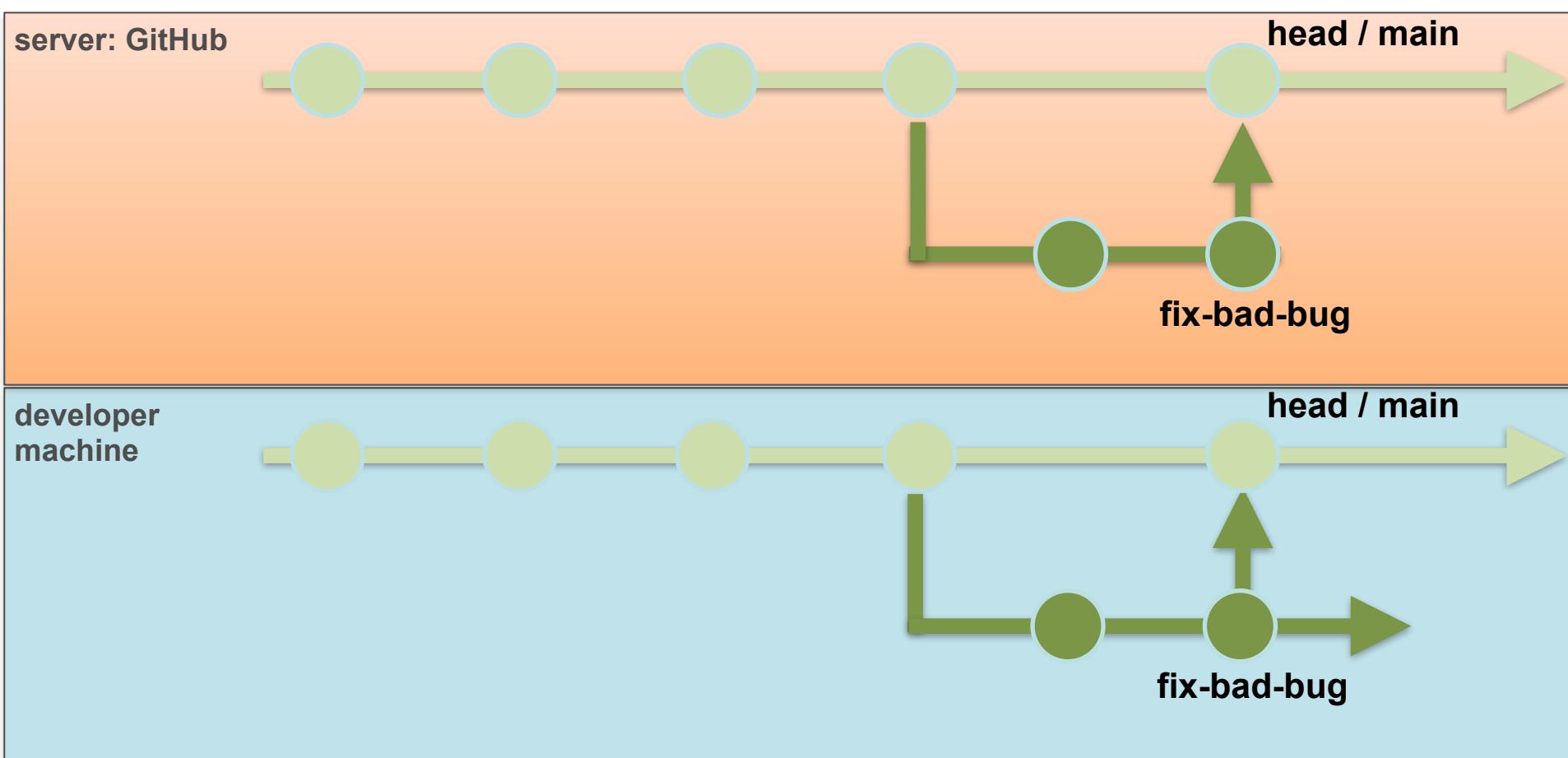


Repos & Branches



merges pull request of **fix-bad-bug** into **main**
// choice of whether to delete **fix-bad-bug** branch or keep it for historic
reasons after merge. Usually deleted because main commit has all info

Repos & Branches



git checkout main // goes back to main locally

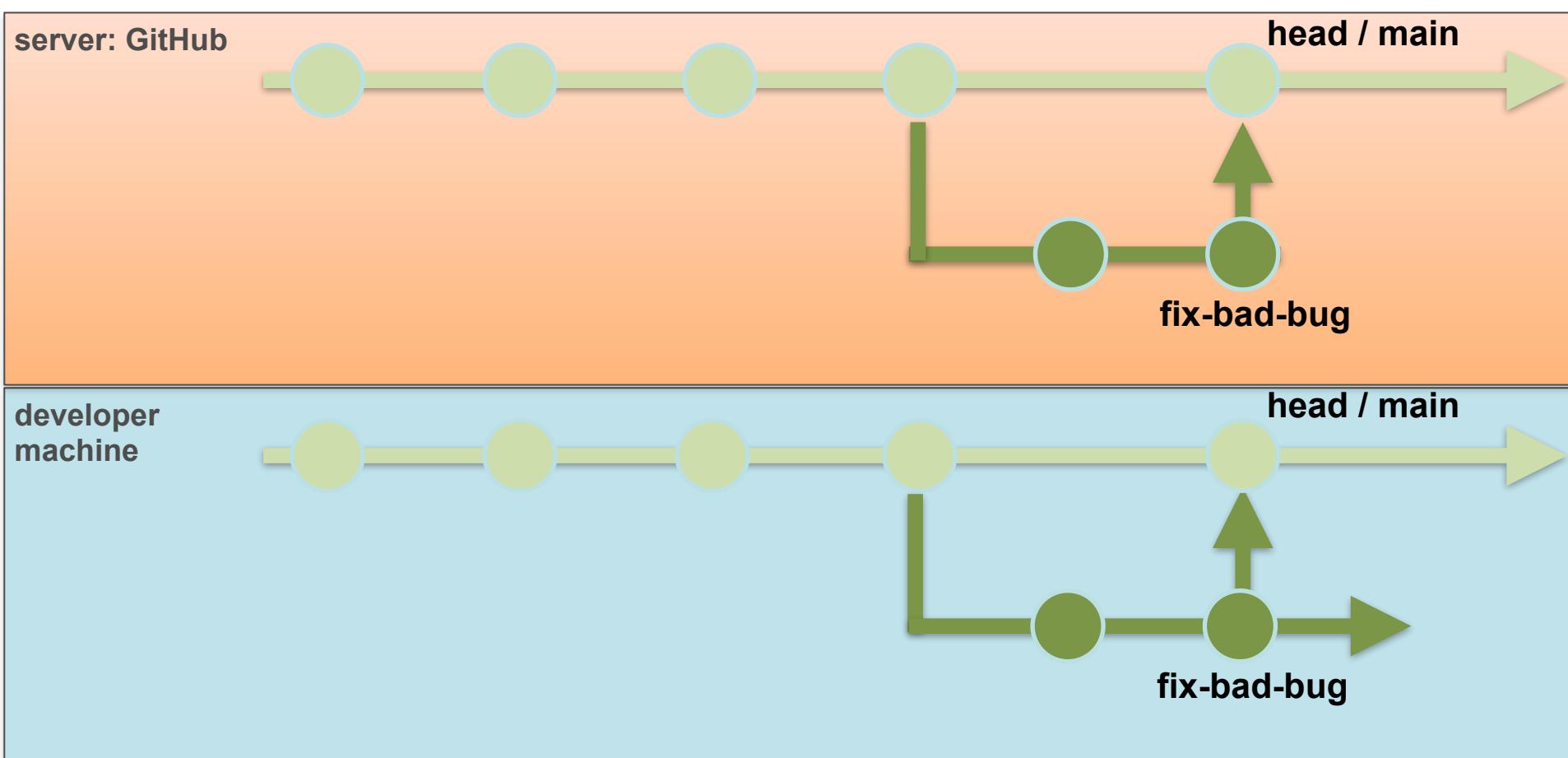
git pull // to get all of latest main changes locally



ctrical & Computer Engineering



Repos & Branches



Rinse and Repeat!!



ctrical & Computer Engineering



Day 1 Hands-on Exercise 0

- Create your own repos based on GitHub template
- Clone the repo locally - **git clone**
- Email me your GitHub usernames now please! - esolovey@bu.edu

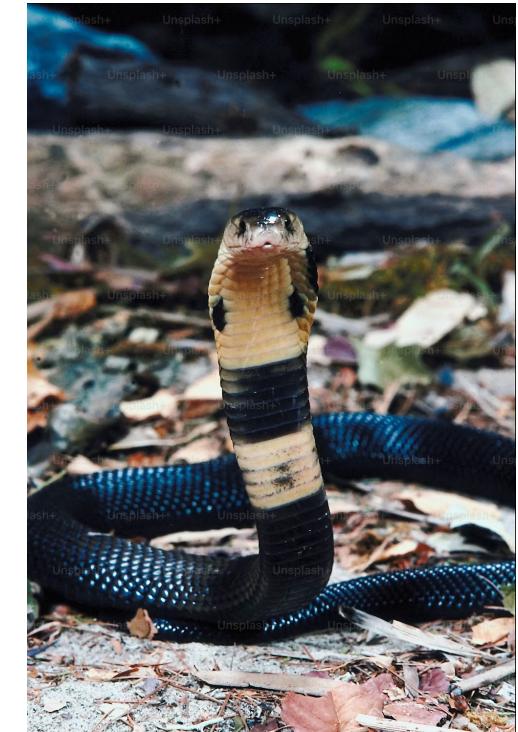


Day 1 Hands-on Exercise 0 - What did you learn?



Day 1 - Agenda

- GitHub
 - Forking base repo and setting up your own repo
- Dissecting a program



```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

# TODO: compute average final score

# TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What do you think will happen when we execute this scaffold?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What can you imply about Python variable typing from this program?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What do you think the indentation level signifies (if anything) ?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What do you think the **scope** of the **reader** variable is?
 - What is **scope!!?**

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

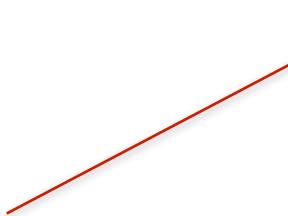
        # TODO: compute average final score

        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What do you think the **scope** of the **f** variable is?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

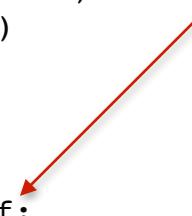
    # TODO: compute average final score

    # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What do you think the **scope** of the **input_filename** variable is?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

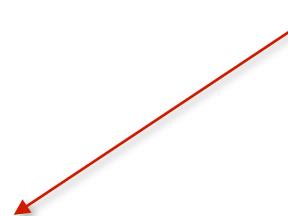
        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```

- What do you think the **scope** of the **scores** variable is?



```
def compute_number_of_unique_students(self, scores: [ExamScore]) -> int:  
    """  
        Given a list of ExamScore's, computes the number of unique students in the data set  
    """  
  
    student_ids = set()  
    for score in scores:  
        student_ids.add(score.student_id)  
  
    return len(student_ids)
```

- What do you think the **result** variable is?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

        # TODO: unique student count

if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



- What are your immediate thoughts on implementing the TODO's ?

```
import csv
import os
import json

input_filename = "test_scores.csv"
output_filename = "output.json"

if os.path.exists(output_filename):
    os.remove(output_filename)

average_final = 0.0
unique_students = 0

with open(input_filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

        # TODO: compute average final score

        # TODO: unique student count

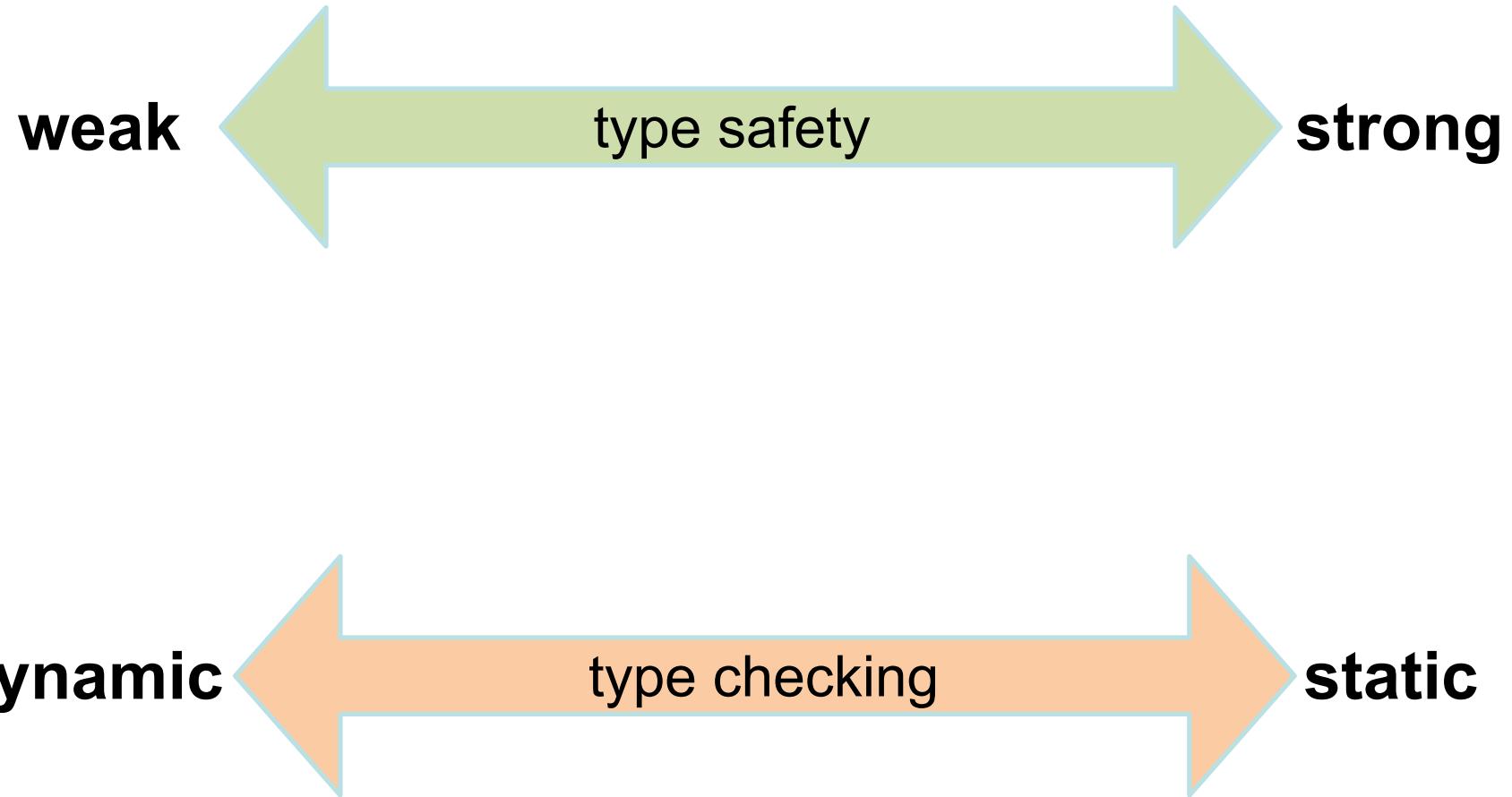
if os.path.exists(output_filename):
    os.remove(output_filename)

result = {
    "average_final": average_final,
    "unique_students": unique_students,
}

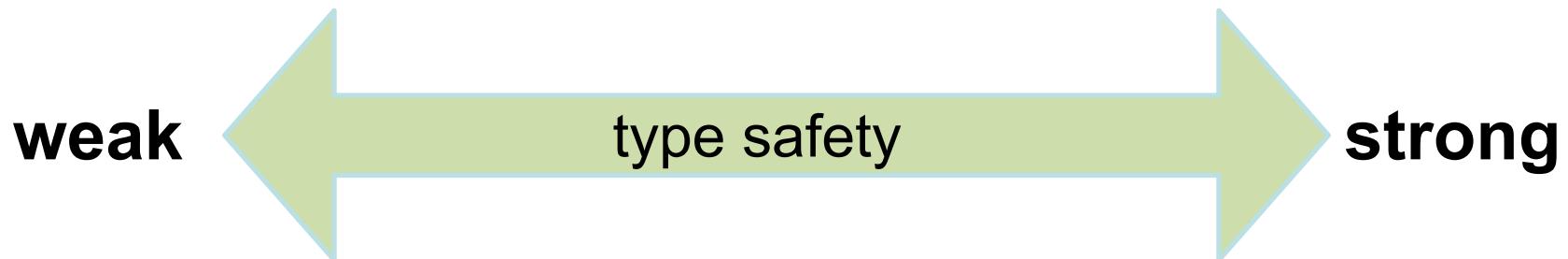
with open(output_filename, "w") as out:
    json.dump(result, out, indent=2)
```



Language Typing

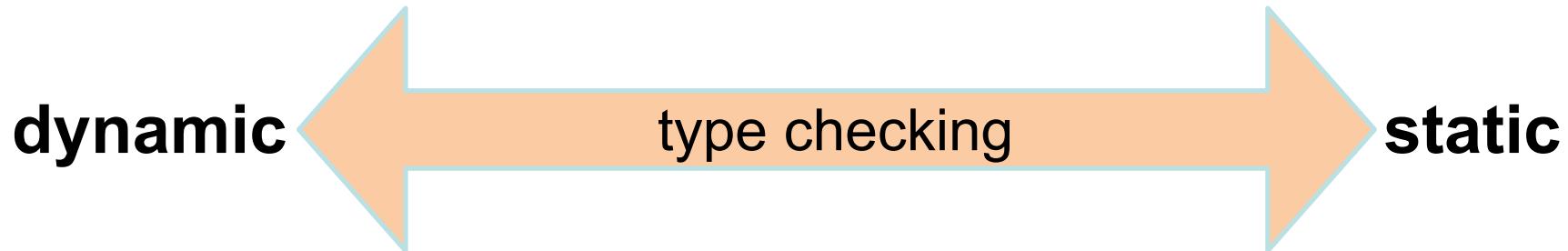


Type Safety



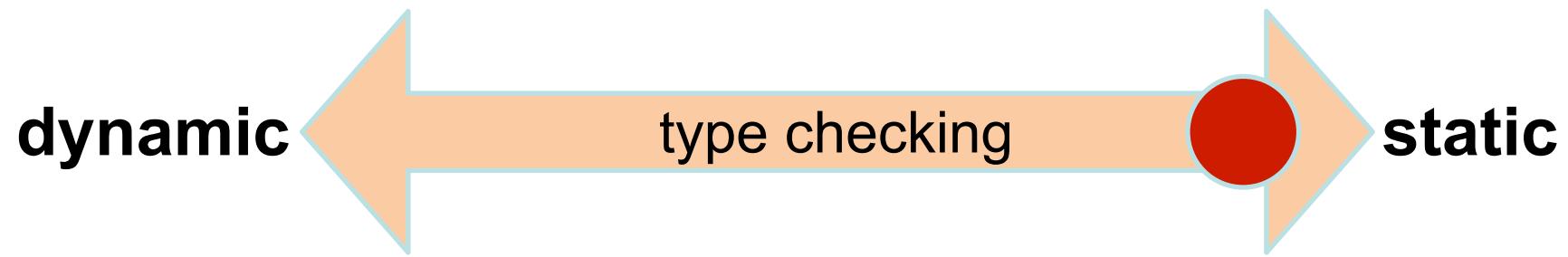
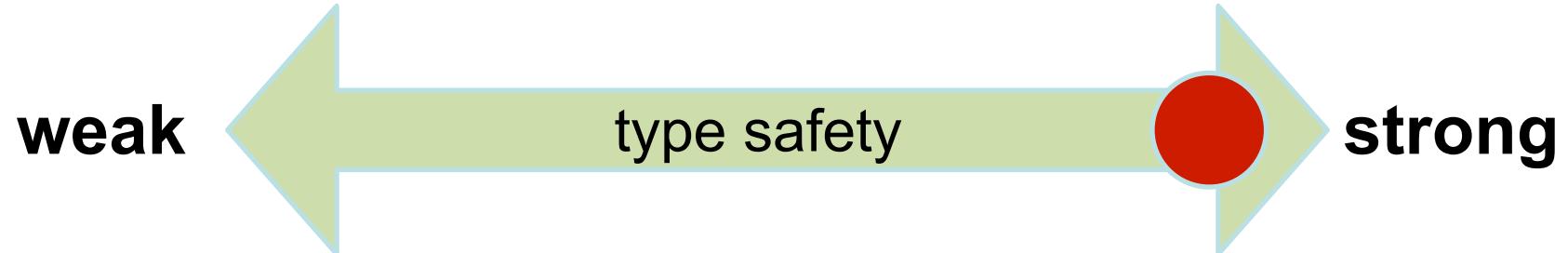
- Type Safety refers to how strictly a language enforces rules about how variables of different types can interact.
- Weakly typed languages have lenient rules where the language attempts to **implicitly** convert and coerce values
- Strongly typed languages have strict rules where unsupported operations between types result in errors.
- Type Safety is very much a spectrum.

Type Checking

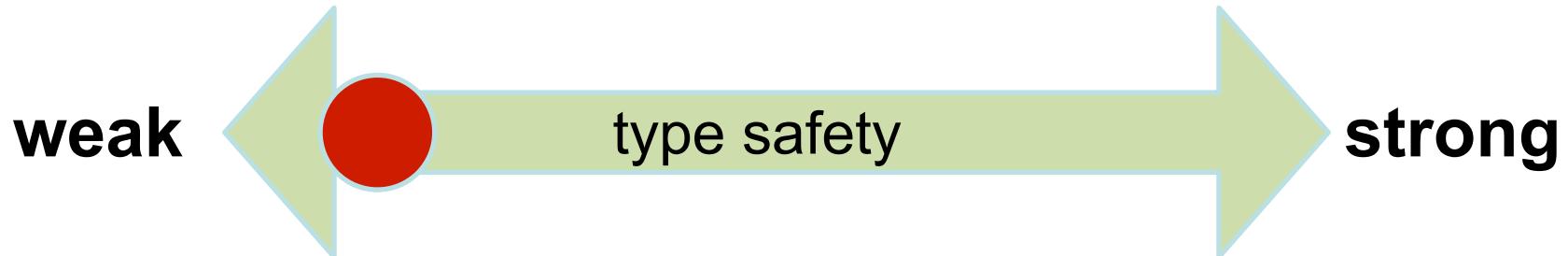


- Type Checking is the process of verifying whether operations are being applied to compatible data types
- Dynamic type checking happens at runtime
- Static type checking happens at compile time
- This is less of a spectrum with most languages landing firmly at one extreme or the other

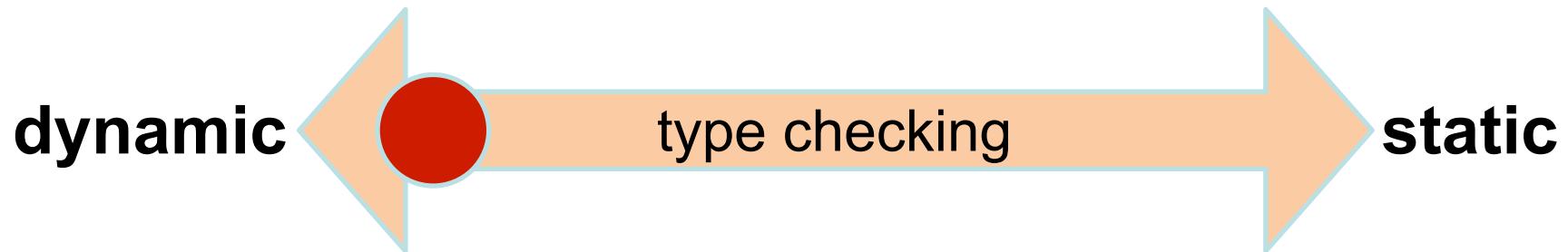
Language Typing : Java



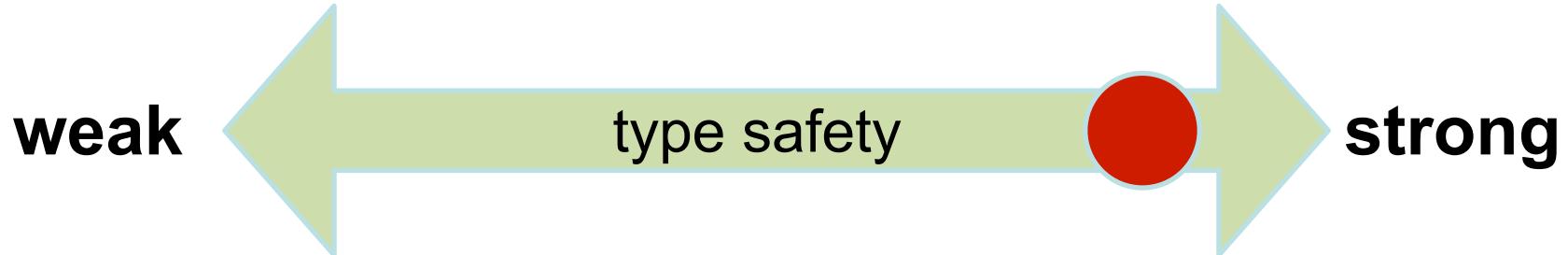
Language Typing : JavaScript



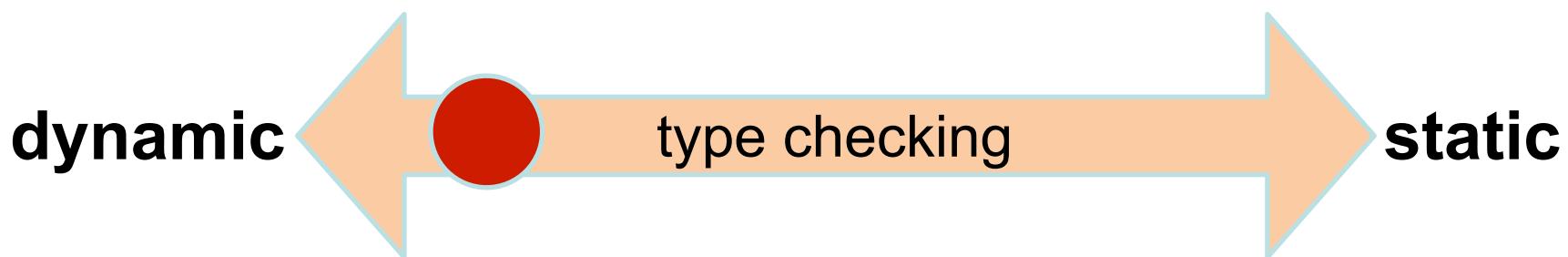
```
console.log("b" + "a" + + "a" + "a");  
baNaNa
```



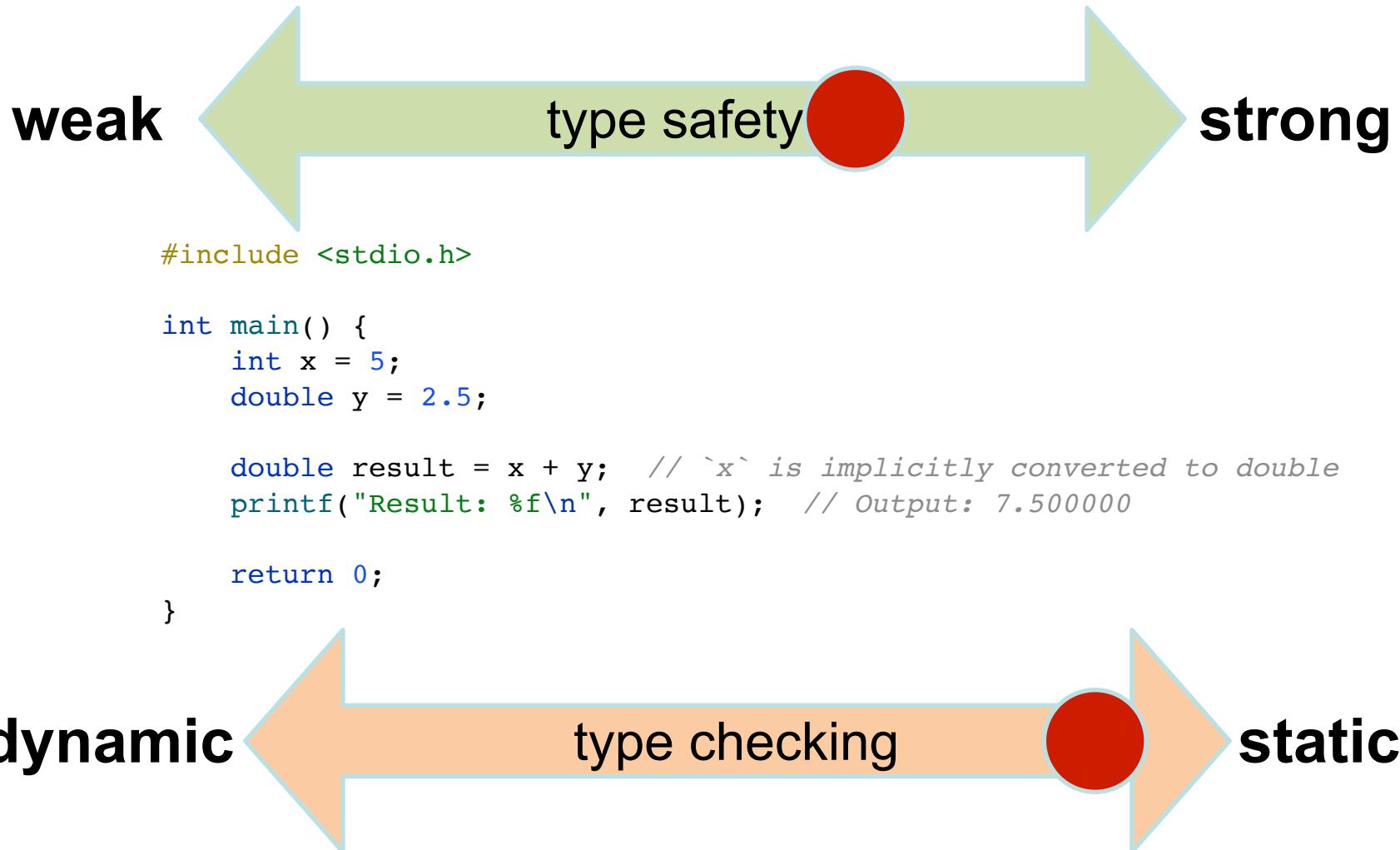
Language Typing : Python



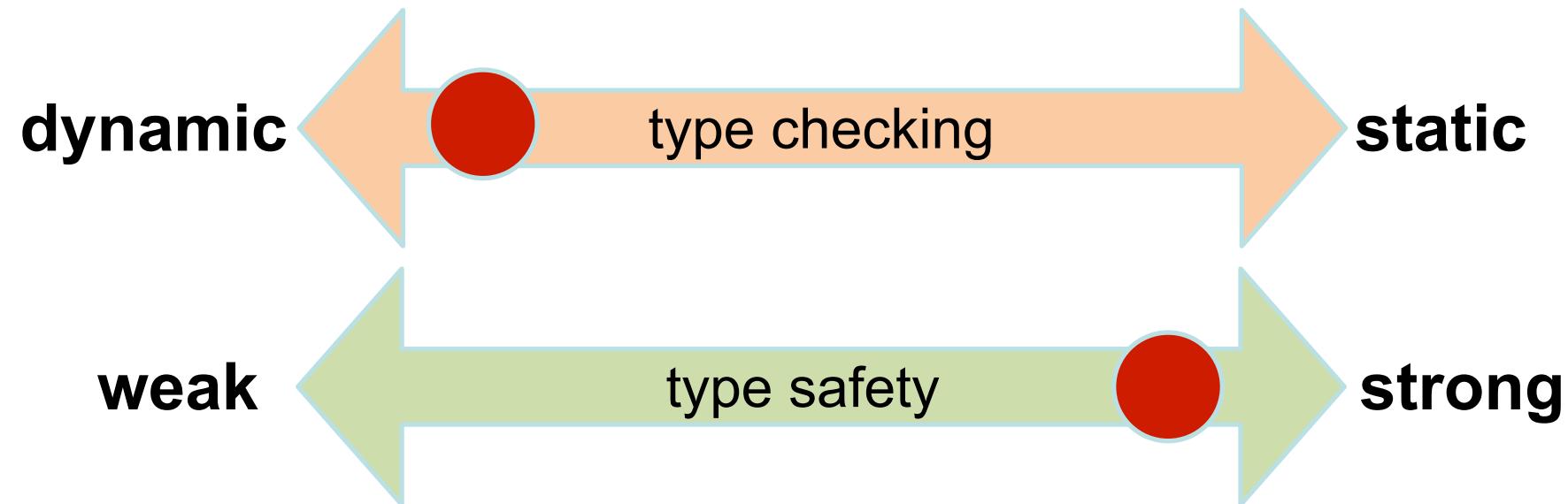
- Variables have no declared type at compile time
- They do have strong types at runtime depending on the values that they hold
- Runtime operations between incompatible types are mostly prohibited and error out



Language Typing : C/C++

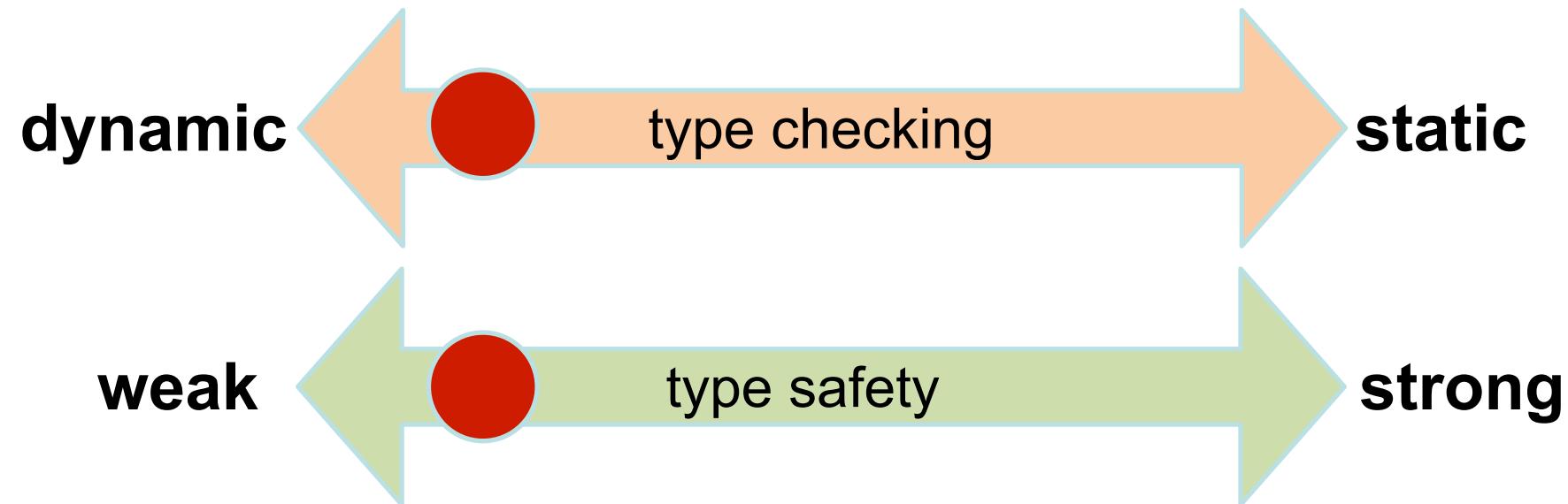


Python



```
x = 10      # int  
x = "hello" # now a str    ← dynamic typing  
  
y = 5  
print(x + y) # TypeError: can only concatenate str (not "int") to str  
              ↑  
              strong typing
```

JavaScript



```
let x = 10;      // int  
x = "hello";    // now a string
```

dynamic typing

```
let y = 5;  
console.log(x + y); // "hello5"
```

weak typing

Day 1 Hands-on Exercise 1

- Set up Python environment and PyCharm
- Run skeletal program that you got as part of the template repository
- Implement the TODOs



1

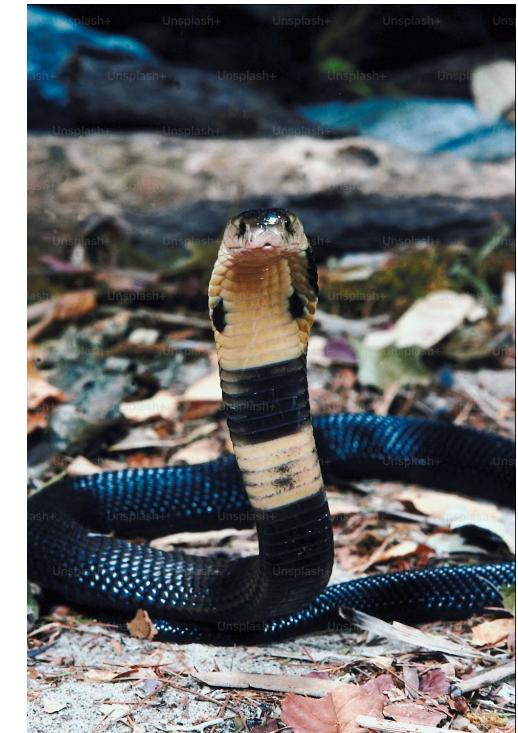
A large, light blue circle containing the number '1' in white.

Day 1 Hands-on Exercise 1 - What did you learn?



Day 1 - Agenda

- GitHub
 - Forking base repo and setting up your own repo
- Dissecting a program
- Installing PyCharm
- **Introduction to testing**

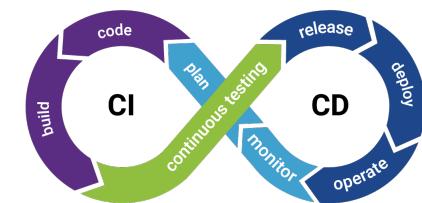


■ Why test software?



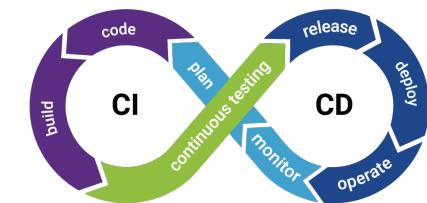
Why do we need software tests?

- Identify bugs/defects before users do
- Maintain system malleability:
 - Gain confidence that system maintains functionality after changes are made:
 - small – incremental fixes/features
 - large – a refactoring
- Enables continuous integration/deployment
 - Will get into detail later, but gives us confidence to ship code frequently (all the time)



Continuous Testing / Integration / Deployment

- These terms get muddled together frequently
- We will talk later in a lot more detail about Continuous Integration and Continuous Deployment, but so you are aware of the distinctions:
 - **Continuous Testing** - running an automated test-suite on every change to a code base.
 - **Continuous Integration** - merging code to your **main** branch very frequently (at least once per engineer per day)
 - **Continuous Deployment** - deploying every change to **main** to your production environment.
- All follow a similar mindset but accomplish different things and require different set up.



Why? - Keep Code Malleable

- Consider needing to make a change to:
 - System with very little test coverage VS
 - System with high test coverage
- What does the latter allow you to do?
 - Can run tests BEFORE and AFTER proposed change and see if any start to fail - **looking for regressions.**
 - Does having a lot of tests guarantee that your change does not introduce a problem?
 - Probably not to 100% but it increases confidence and we can push towards that goal



Why? - Reduce Bugs Found by Customers

- We want to keep our users/customers happy and excited to use our products - every bug they encounter reduces that happiness a bit
- Will a coherent test plan and existence of tests drive the number of bugs users find to zero?
 - Probably not... but we can move towards that goal and will explore techniques in addition to tests that will help

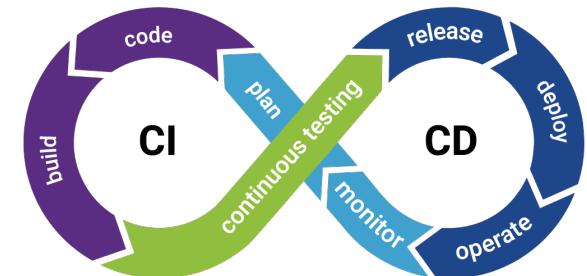


software users



Why? - Continuous Deployment

- Will talk in a lot more detail later, but contrast:
 - Decades ago: ship software to customers on a disk
 - Software as a Service: full control of software that a customer experiences
- As code delivery paradigm changes, testing philosophy changes

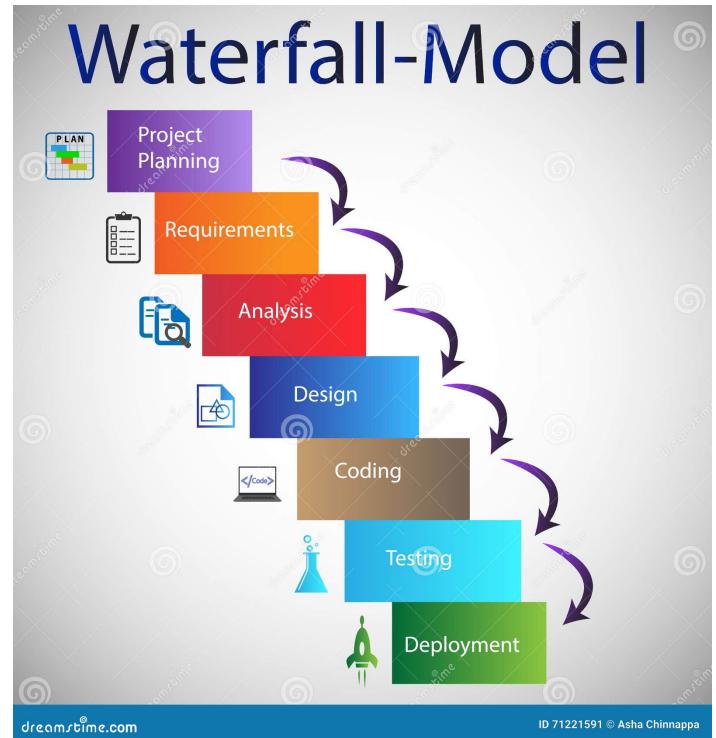


- Why test software?
- Who tests the software?



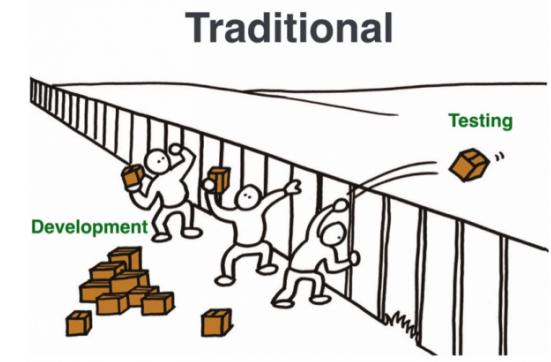
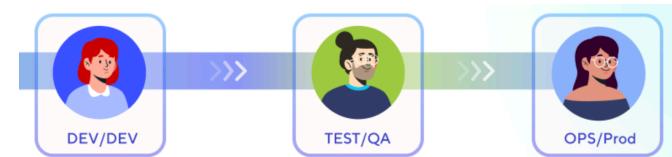
Who is Testing the Software?

- Historically, people who wrote the software were not the ones testing it. For many reasons:
 - waterfall approach
 - segment phases of software lifecycle
 - coders would move on to next feature while quality assurance tested previous one



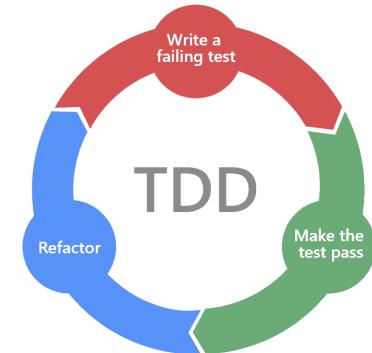
Who is Testing the Software?

- Segmentation of phases also led to segmentation of roles / skills
 - emphasis on manual testing
 - this led to many negative consequences
 - lack of collaboration
 - miscommunication
 - lost opportunities for efficiency and learning



Who is Testing the Software?

- In virtually all modern software development environments, the same people are writing & testing the software
 - Tighter coupling helps ensure that features are not “done” until they are tested
 - Opportunities for taking advantage of knowledge about how code is written are utilized when thinking about how it is tested
 - Opportunities for taking advantage of knowledge about how code might be / is tested are not lost when thinking about how it is written

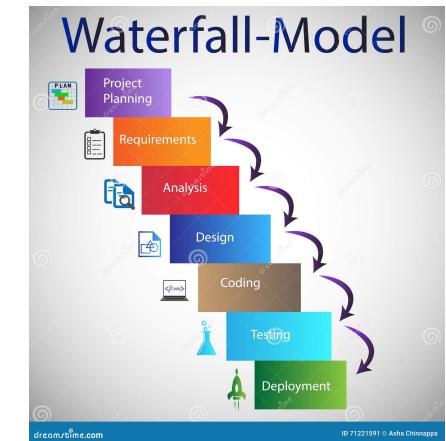


- Why test software?
- Who tests the software?
- **When is the software tested?**



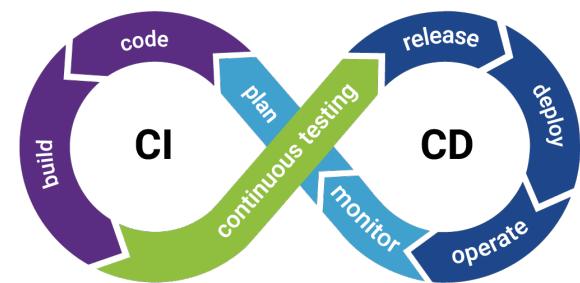
When is the Best Time to Test the Software?

- Historically:
 - testing happened in a dedicated phase by dedicated people
 - Intensely right before shipping software



When is the Best Time to Test Software as a Service?

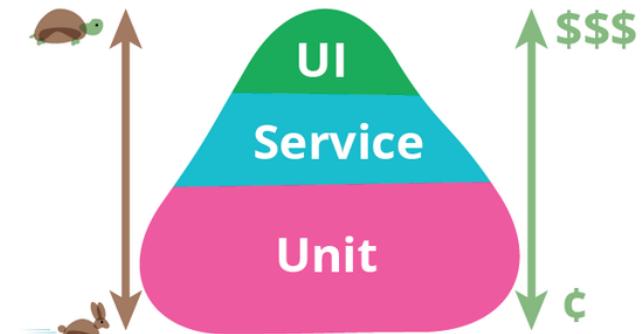
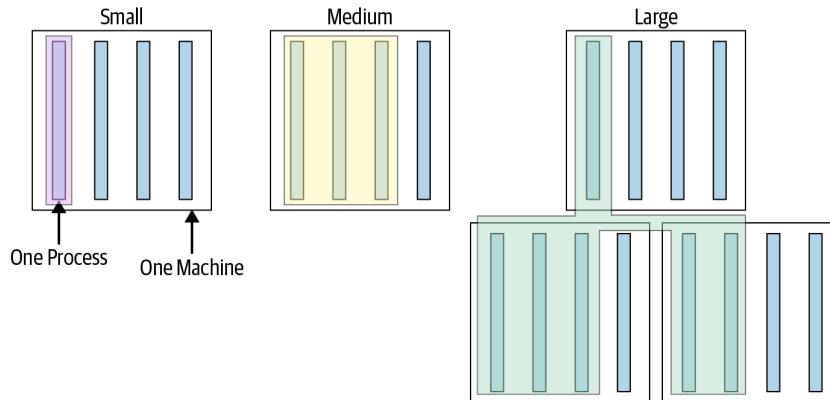
- All the time! Continuously ...
 - via automated test suites that can be run by developers locally
 - via automated test suites that allow for Continuous Integration when multiple developers are working in the same code base
 - via automated test suites that Continuous Deployment servers can run before deploying code to production servers



- Why test software?
- Who tests the software?
- When is the software tested?
- **What is tested?**



What does each test focus on?

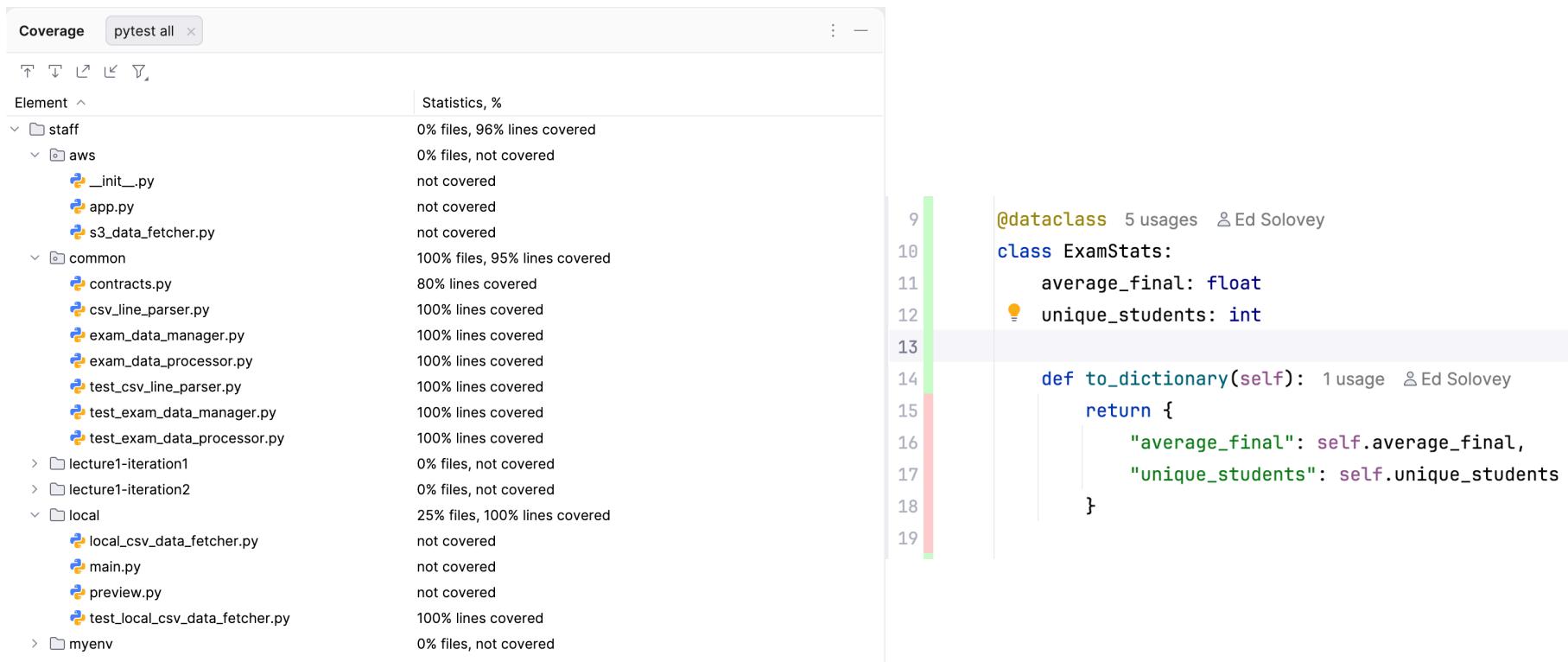


- **Unit = Small**
 - fast
 - deterministic
 - cheap
 - focused
- **Service = Medium**
 - slower
 - less deterministic
 - less cheap
 - broader
- **UI/End-toEnd = Large**
 - slow
 - least deterministic
 - expensive
 - broadest

How do we know when we have added enough tests?



Code Coverage



Example Test

```
import json
import os
import subprocess
import sys
from pathlib import Path

def test_script_output():
    repo_root = Path(__file__).resolve().parents[1]
    print("repo: " + str(repo_root))
    output_filename = "output.json"

    result = subprocess.run(
        [sys.executable, "-m", "main"],
        cwd=str(repo_root),
        capture_output=True,
        text=True
    )

    assert result.returncode == 0, (
        f"Return code: {result.returncode}\n"
        f"cwd used: {repo_root}\n"
        f"STDOUT:\n{result.stdout}\n"
        f"STDERR:\n{result.stderr}\n"
    )

    assert os.path.exists(output_filename)

    with open(output_filename) as f:
        main_output = json.load(f)

        assert "average_final" in main_output
        assert "unique_students" in main_output

        assert main_output["average_final"] == 87.33
        assert main_output["unique_students"] == 3
```



Day 1 Hands-on Exercise 2

- Install Pytest
- Write unit tests for your current implementation without re-factoring it
 - Get confidence that it works for edge cases
- Modify your `test_main.py` to look like the version here - https://github.com/BU-2026-Leap/base/blob/main/test_main.py
- To ensure input file can still be read when invoked from test, modify `main.py` to look like (this change is available in [main.py of the base repo](#)):

```
from pathlib import Path

full_base_path = Path(__file__).resolve().parent
input_filename = full_base_path / "test_scores.csv"
```

- Set up Continuous Integration via GitHub Actions

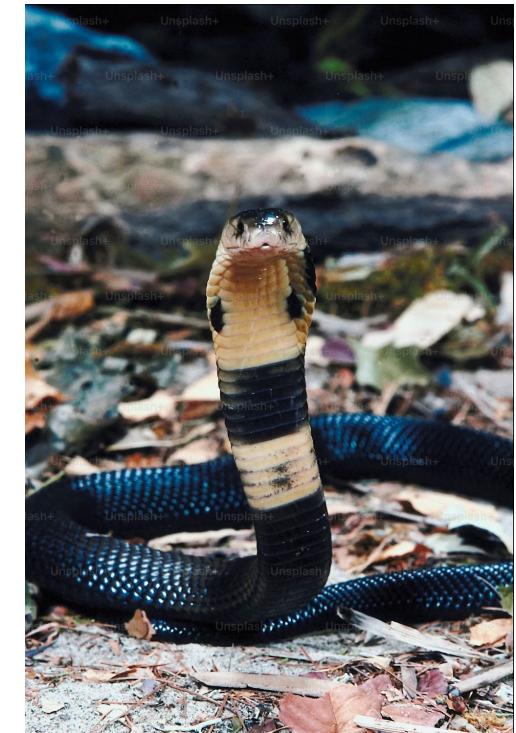


Day 1 Hands-on Exercise 2 - What did you learn?



Day 1 - Agenda

- GitHub
 - Forking **base** repo and setting up your own repo
- Dissecting a program
- Installing PyCharm
- Introduction to testing
- Continuous Integration (via GitHub Actions)
- **Re-factoring to improve testability**
- **Re-factoring to improve readability, code organization**



Structure Your Code Base For Ease of Extension

- “For each desired change, make the change easy (warning: this may be hard), then make the easy change.” - Kent Beck



Kent Beck

[Article](#) [Talk](#)

From Wikipedia, the free encyclopedia

Kent Beck (born 1961) is an American [software engineer](#) and the creator of [extreme programming](#),^[1] a [software development methodology](#) that eschews rigid formal specification for a collaborative and iterative design process. Beck was one of the 17 original signatories of the [Agile Manifesto](#),^[1] the founding document for [agile software development](#). Extreme and Agile methods are closely associated with [Test-Driven Development](#) (TDD), of which Beck is perhaps the leading proponent.



Department of Electrical & Computer Engineering



Structure Your Code Base For Ease of Extension

- “For each desired change, make the change easy (warning: this may be hard), then make the easy change.” - Kent Beck
- One way to make changes easy is by structuring our code base such that we can **extend our system by adding code rather than changing existing code**



Structure Your Code Base For Ease of Extension

- **Greenfield project** - developing a system for a totally new environment, without concern for integrating with other systems, especially not legacy systems.
- Engineers often pine for greenfield projects instead of working with legacy code
- If we structure our code thoughtfully, then extensions to existing code bases will feel like greenfield projects!



What does “Good” mean in the context of “Good” Code

- **Maintainability**
 - How easy is it for someone on your team to fix a bug or add a new feature?
- **Testability**
 - Can tests that themselves are maintainable be written at all levels?
- **Readability**
 - How easy is it for someone who has not looked at the code ever, or in a few months, to understand it?
- **Malleability**
 - When you discover new things about your customer or product needs, or have a new technical idea, is it straightforward to re-organize the code to suite those needs?



“Good” Code at a High Level

- Organization: where does the code go?
 - Which class? method? How many classes do we have?
- Interactions: what are the connections between various pieces of code?
 - Does each class depend on all others? just a few?
- Patterns
 - Is readability improved by following commonly known patterns?



“Good” Code at a Lower Level

- How is state represented?
 - Is it well encapsulated?
 - Is it mutable? Overly mutable?
- What does the code look like?
 - What are the variable, method names like?
 - What is the spacing like?
 - Is the code consistent from class to class?



High Level Concepts

- **Modularity**



Modularity

- A coherent component
 - has a **Single Responsibility**
 - as few dependencies on other modules as possible
 - **Encapsulates State** and implementation - publicly exposes as little as possible
 - is testable
 - is reusable
- Components have a **Separation of Concerns**
 - no overlap of responsibilities



Single Responsibility Principle

- Each component/module should do exactly one thing
 - Do it well
 - Do nothing more than that one thing
- For example, consider a case where you are building a back-end for an application that needs to be able to
 - Create users in your system
 - Notify users of events in your system
 - Successful creation being one of them



Encapsulation

- In context of Object Oriented languages, like Java, has two aspects:
 - State and logic that manipulates that state should live in the same Object.
 - The state of the Object should be manipulated by it and not permitted to be manipulated by others



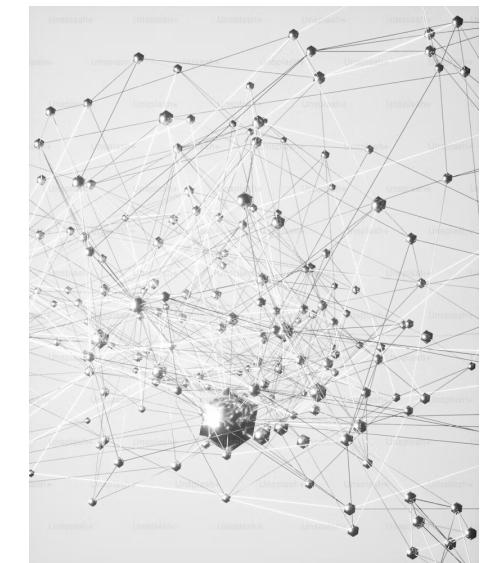
Modularity

- **Single Responsibility Principle** and **Encapsulation** can be thought of independently of **Modularity**, but they such crucial components of it that it makes sense to consider them alongside of it.



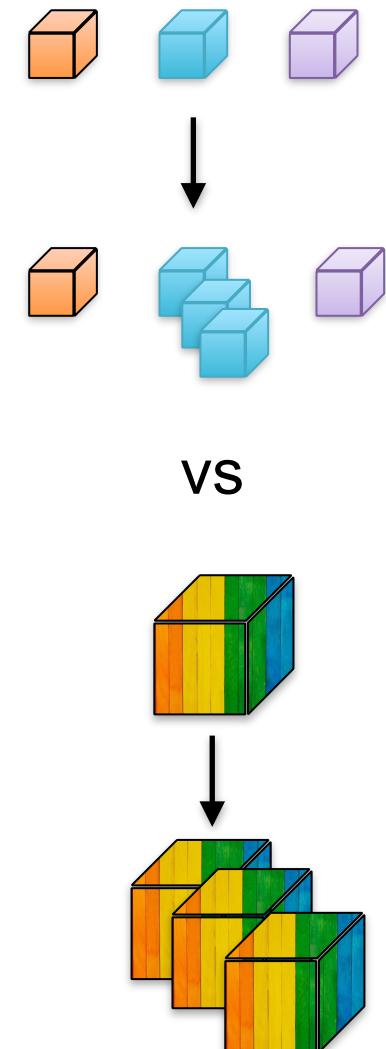
Coupling (opposite of Modularity)

- No coherent components/modules
- Or every component/module depends on every other component/module
- Reduced
 - readability
 - cohesion
 - encapsulation
 - maintainability
 - testability



Modularity vs Coupling

- Might be the most important distinction in software engineering
- Impacts every other aspect of your system
- Scaling a tangled, coupled system is virtually impossible
 - Need to ramp up a single component to meet demand but need to deploy more of the entire system because it is too coupled
 - extremely expensive
 - and inefficient



vs

High Level Concepts

- Modularity
- **Simplicity**



Simplicity

- Think about the problem you are trying to solve and come up with as simple solution as possible for that exact problem - not an adjacent one, not a potential evolution of it in the future.
- **Avoid Over-Engineering**
 - Do not design a system that is more complex than necessary
 - Do not build features or capabilities that are not needed or are unlikely to be used.
 - Resist the urge to build for potential future needs that may never materialize. Focus on solving the current problem effectively rather than anticipating every possible future scenario.



Simplicity

- The more code we have in a system the higher the maintenance cost (even if it is the most beautiful code) - implementing things that our users do not need (yet) is expensive for many reasons:
 - readability
 - testability
 - maintainability
- We will go deeper into building just what we need **now** when we talk about Engineering Processes



What is the definition of a SINGLE module?



Day 1 Hands-on Exercise 3

- Re-factor your code such that core computations happens in isolated, dedicated functions
 - Write unit tests just for those functions
- Confirm that your code still works and that your Continuous Integration still passes



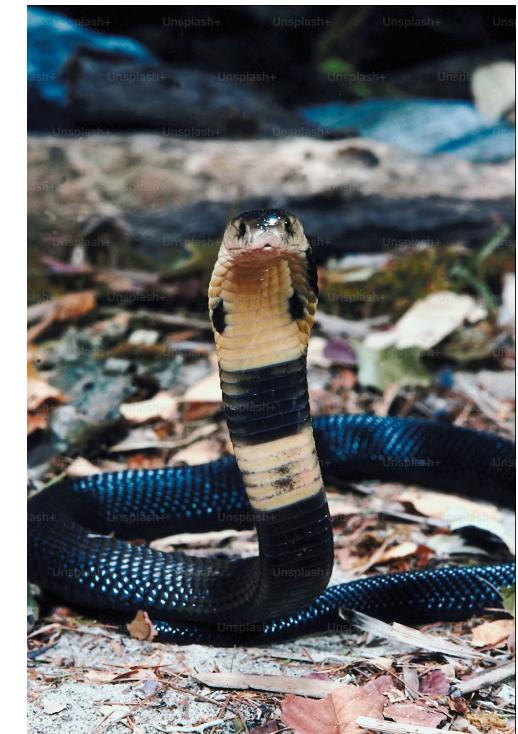
3

Day 1 Hands-on Exercise 3 - What did you learn?



Day 1 - Agenda

- GitHub
 - Forking **base** repo and setting up your own repo
- Dissecting a program
- Installing PyCharm
- Introduction to testing
- Re-factoring to improve testability
- Continuous Integration (via GitHub Actions)
- Re-factoring to improve readability, code organization
- **Object Oriented (Basics) to improve code organization**



What are the benefits of OOP?

- **Encapsulation**

- Briefly talked about this best practice principle during last lecture.
 - Prior to OOP, data could be encapsulated
 - OOP enables encapsulation of the data as well as its supported behaviors
- This is particularly beneficial for larger code bases as it makes them more
 - modular
 - readable
 - maintainable
 - scalable

- **classes** are templates that define state (variables) and methods (functions) that are tightly coupled together.
- **instances** are specific objects created from the **class** template that have their own state based on the **class** definition.
 - Your program may involve very many instances of a particular class, just as it may involve many **integer** or **string** variables.
- A lot more core concepts to come but let's start on an example that demonstrates these two most core ones

```
class ExamScore:  
    student_id: str  
    exam_name: str  
    score: float
```



three pieces of information
encapsulated together

```
class ExamScore:  
    student_id: str  
    exam_name: str  
    score: float
```

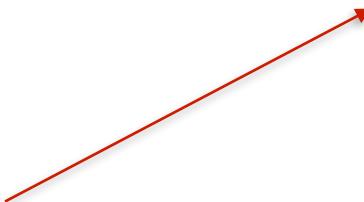
creation of an **instance** of the **ExamScore** class

```
expected = ExamScore(  
    student_id="U123",  
    exam_name="midterm",  
    score=75.4  
)
```

```
@dataclass
class ExamStats:
    average_final: float
    unique_students: int

    def to_dictionary(self):
        return {
            "average_final": self.average_final,
            "unique_students": self.unique_students
        }
```

behavior (converting to a dictionary)
encapsulated together with state



```
@dataclass
class ExamStats:
    average_final: float
    unique_students: int

    def to_dictionary(self):
        return {
            "average_final": self.average_final,
            "unique_students": self.unique_students
        }
```

invocation of a method (behavior)
on an instance of the **ExamStats**
class

```
result = read_and_compute(
    LocalCSVDataFetcher(INPUT_FILENAME),
    ExamDataProcessor()
)

with open(OUTPUT_FILENAME, "w") as out:
    json.dump(result.to_dictionary(), out, indent=2)
```

```
from dataclasses import dataclass

@dataclass
class ExamScore:
    student_id: str
    exam_name: str
    score: float

@dataclass
class ExamStats:
    average_final: float
    unique_students: int

    def to_dictionary(self):
        return {
            "average_final": self.average_final,
            "unique_students": self.unique_students
        }

class DataFetcher:
    def fetch(self) -> [ExamScore]:
        raise NotImplementedError

class DataProcessor:
    def compute_average_final(self, scores: [ExamScore]) -> float:
        raise NotImplementedError

    def compute_number_of_unique_students(self, scores: [ExamScore]) -> int:
        raise NotImplementedError
```

Day 1 Hands-on Exercise 4

- Re-factor your code to use classes
- Re-factor your code to implement interfaces
- Update your tests to cover your updated structure
- Make sure your code still works



4

Day 1 Hands-on Exercise 4 - What did you learn?



Notes

- Go over effective use of LLMs
- Explain that will go over concepts faster than we can explain them, but will circle back to them
- Talk about
 - typing (strong vs weak)
 - typing (dynamic vs compiled)
 - scope
 - code readability
 - classes
 - TDD
 - benefits of focused tests
 - have test fail and show how we would fix
- Talk about collection comprehensions
- parse local csv then parse json over http?