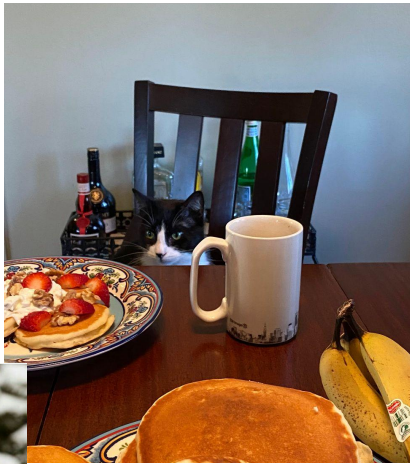
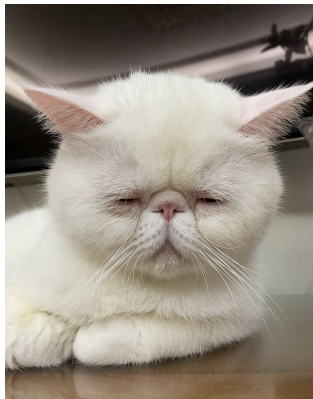
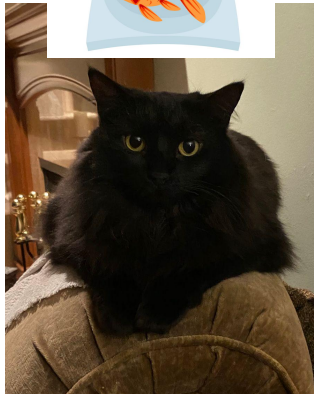


EECS 388: Lab 1

Introducing Project 1
Project Mechanics
Python Overview



Welcome to EECS 388 Lab!



Participation

- Goal: Intellectual Contributions
 - asking good questions
 - making thoughtful comments
 - helping others appropriately
- Many ways to get points
 - Piazza
 - Lab
 - Lecture
- We'll take attendance in lab to help us get to know you, but showing up or not doesn't affect your score.
What does is intellectual contributions.
- Contributions on Piazza can also earn you full points.



Course Projects

Course Projects

- Five projects = 45% of your semester grade
- Labs will introduce you to the projects and help guide you
- Projects have multiple parts - use time wisely
 - ***Seriously, start early!*** Come to office hours, in person or online
 - **3 late days for all routine illness, tech problems, scheduling conflicts, etc.**
- Project 1–4: Work **individually or with a partner**
 - May switch partners between projects, but not within a project
- Project 5: Must work with **a partner**.
- For each project, a simple **Lab Assignment** will help introduce you to languages and tools (total of 5% of semester grade)
- Make sure you have **30 gigabytes** of free space for projects.
 - If you don't, Google Drive or a USB Stick could help you make space
 - If you don't have compatible hardware, reach out to course staff

Project Preview: Security Tools and New Languages

1. Cryptography - Python

- Length extension attack, hash collisions, RSA signature forgery, padding oracle attack

2. Web - SQL/Javascript

- Database (SQL) injection attacks, cross-site scripting attacks, cross-site request forgery

3. Networking

- Network packet analysis, network attacks and defenses



4. Application Security - C/Python

- Buffer overflow attacks, reverse engineering



5. Forensics

- Disk image analysis, steganography



*assignments are due at 6 p.m., not midnight!

Project Expectations

- Projects are a deep-dive in specific topics and techniques
 - *Not everything you'll need to know is taught in class—and that's to be expected!*
 - Specs will point to further reading and research
 - Sometimes you may need to pick up new programming languages: that's expected at your level of CS education
 - Labs will give an initial walkthrough of new languages and particular tools
- You are bound by the Honor Code
 - Google is also your friend for finding reference material (but ***not solutions!***) (Not sure if a source is OK to use? Ask us!)
 - See the course site for details about our policy on collaboration
 - Violators will be reported to the Honor Council
 - You're not allowed to use hints/solutions/code from others (including from an AI system)




Getting Help

Piazza

- <https://piazza.com/umich/fall2024/eecs388>
- Please be polite and first check if your question was answered already :-)
- Also, spread the knowledge by answering your fellow students' questions!
 - This will contribute to your participation!

Office Hours

- Online and in-person. Schedule posted at eecs388.org (subject to change)
 - You can also add our calendar to yours: Go [here](#)
 - Join the queue: [in-person](#) or [online](#)
 - Please attempt a solution before asking for help, and tell us what you've tried.
- 

Project 1: Cryptography

Investigate vulnerable applications of cryptography,
inspired by problems in many real-world implementations

Lab 1: Due Sep. 5 (6 p.m.)

Part 1: Due Sep. 12 (6 p.m.)

1.1 Length extension

1.2 Hash collisions

Part 2: Due Sep. 19 (6 p.m.)

2.1 Padding oracle

2.2 RSA signature forgery



Project Mechanics

GitHub

Used to distribute starter code for each project

- You will generate a Git repository for each project
- Make sure you keep these repositories **private**
 - Even after the semester finishes
 - *Even after you graduate*



Docker

Used to manage environments for each project

- Container Framework → Allows for a virtual sandbox to run in the background
- Runs a Docker image that will set up the container and environment
- Works seamlessly with VS Code

Further reading and installation instructions are on the [course website](#)



Docker Walkthrough - Installation

1. Windows and macOS: navigate to the [Docker Desktop](#) download page.
 - a. Windows → .exe corresponding to your system
 - b. macOS → .dmg corresponding to your system
 - c. Linux → Docker provides for multiple distributionsInstall Docker Engine, **not Docker Desktop**.
[Docker Desktop currently has a bug with permissions.](#)
2. Run the download and install Docker
3. Once finished installing, open a terminal and run:

```
docker run -it hello-world
```

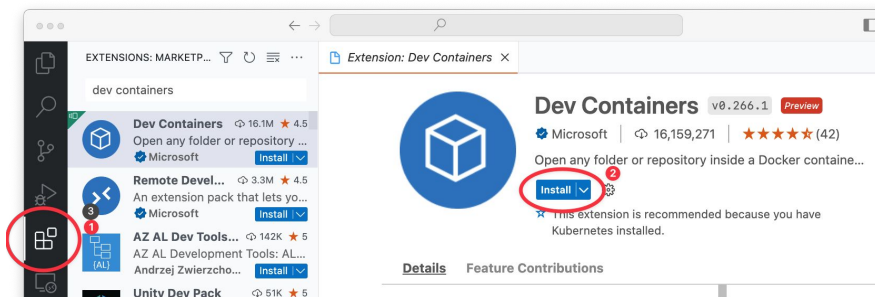
Docker has run a simple program in an *isolated container in the background*



Docker Walkthrough - Visual Studio Code

If you do not have Visual Studio Code already installed, visit <https://code.visualstudio.com/>

1. Open Visual Studio Code.
2. Navigate to “Extensions” on the sidebar.
3. Search for “Dev Containers”, and install it.



We will go into further detail on how to connect this to projects in later labs.

Autograder

Submission platform hosted on autograder.io like EECS 280

- Each project will have a designated list of files to submit for grading
- Make sure to submit early as Autograder can be bogged near deadlines
- Please do not use it as a debugger!



Git

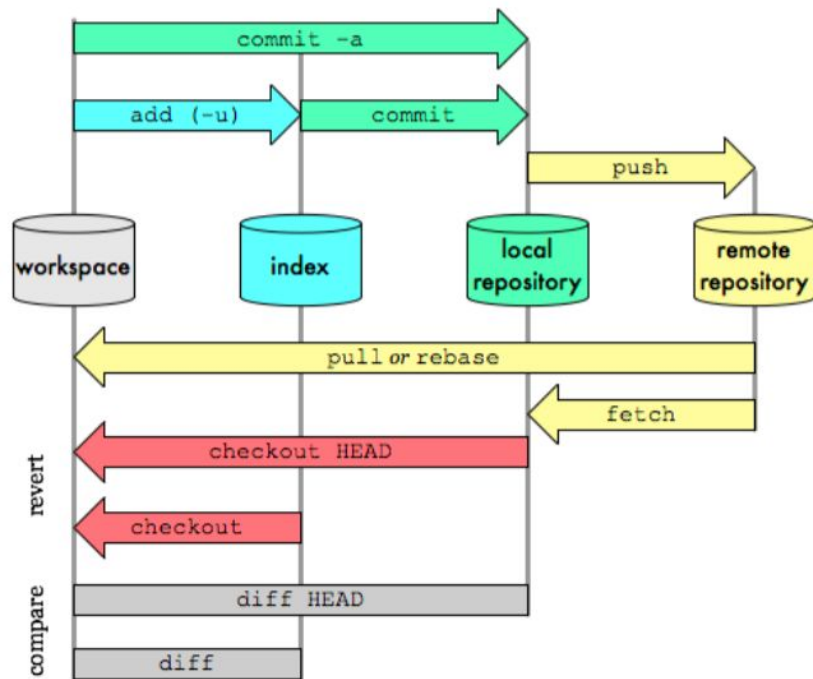


- What is it?
 - Version control system
 - Track file changes and coordinate work among multiple contributors
 - Widely used in industry
- Why do I need it for 388?
 - Synchronize work with your partner (when you have one)
 - Undo mistakes

Quick refresher: <https://www.atlassian.com/git>

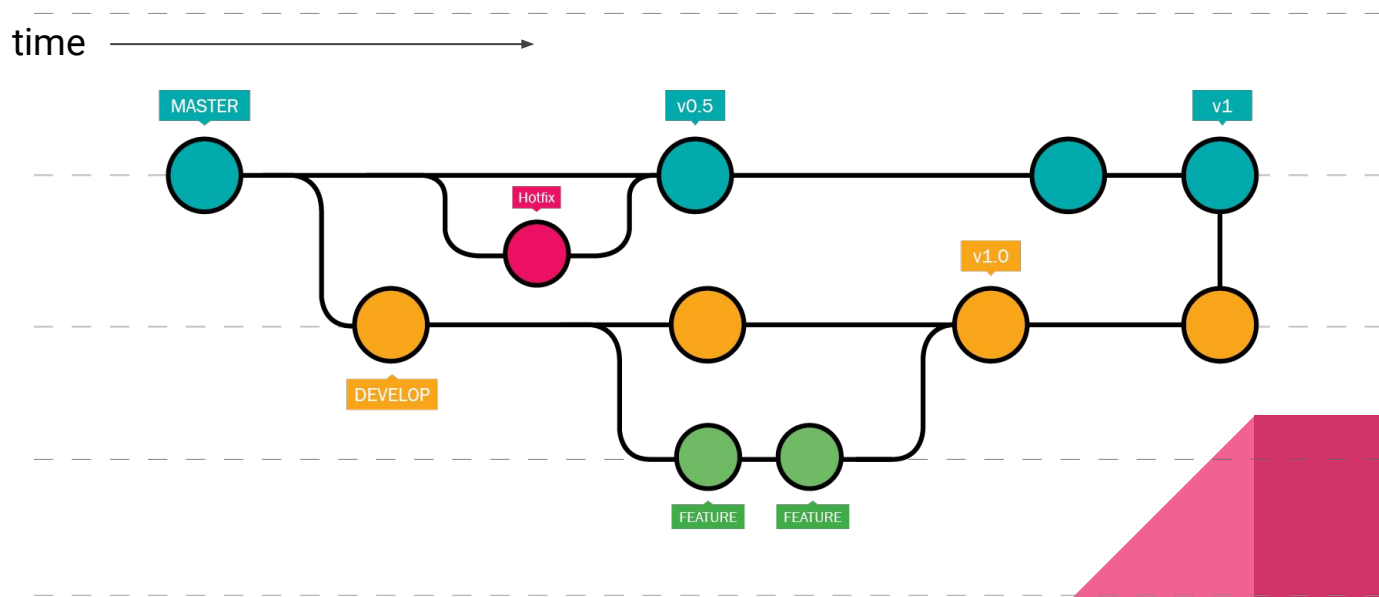
Git Lingo

- **Repository (repo):** project directory, including revision history
- **Clone:** make a local copy of your repo
- **Pull:** Update your *local* repo with any changes made to the *remote* repo
- **Commit:** Save your changes
- **Push:** Update the *remote* repo with changes you made *locally*



Git Lingo

- **Branch:** A parallel version of your repository



Python Introduction

Useful for Project 1!

Hello, world!

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

```
$ python3 filename.py
```



Variables

```
n = 42
```



Functions

```
def add_one(input):  
    return input + 1
```



Tuples

```
def add_and_subtract(a, b):  
    sum = a + b  
    difference = a - b  
  
    return sum, difference  
  
s, d = add_and_subtract(8, 3)
```



Lists

```
l = [1, 2, 3, 4]
l.append(5)

print(l)      # Prints [1, 2, 3, 4, 5]
print(l[0])   # Prints 1
print(l[-1])  # Prints 5
print(l[1:4]) # Prints [2, 3, 4]
```



Dictionaries

```
d = {  
    'eecs': 388,  
    'time': '10:30',  
}
```

```
d['projects'] = 5
```

```
print(d)
```

```
print(d['eecs'])
```



If-statements

```
if n > 0:  
    print('Positive')  
elif n < 0:  
    print('Negative')  
else:  
    print('Zero')
```



For Loops

```
for i in range(5):  
    print(i)
```

```
for i in range(5, 10):  
    print(i)
```

```
l = ['hello', 'world']  
for item in l:  
    print(item)
```

```
d = {  
    'eecs': 388,  
    'time': '10:30',  
}  
  
for key, value in d.items():  
    print(key, value)
```



While Loops

```
while n != 1:  
    print(n)  
  
    if n % 2 == 0:  
        n = n // 2  
    else:  
        n = 3 * n + 1
```



Notable Features

- Control-flow based on whitespace, not brackets
- Only two scopes: local (within the function) and global
- Duck-typing (variable types are figured out at runtime)
- Easy array indexing:

```
$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> msg = "hello"
>>> msg[0]
'h'
>>> msg[-1]
'o'
>>> msg[1:4]
'ell'
>>> msg[:4]
'hell'
>>> msg[-3:]
'llo'
```

Reference Semantics

- All variables in Python are *references* to values, rather than values themselves
 - Kind of similar to pointers in C/C++, but not as scary
- Assigning to a variable (using the = operator) makes a copy of the reference
 - [Visualized Example in PythonTutor](#)
- Function parameters are copies of the references
 - Be careful when writing functions that modify their inputs
 - [Example of accidentally modifying a variable](#)
 - [Example of accidentally not modifying a variable](#)



Working with bytes

[Check the documentation!](#)

Bytes Objects

Bytes objects are immutable sequences of single bytes. Since many major binary protocols are based on the ASCII text encoding, bytes objects offer several methods that are only valid when working with ASCII compatible data and are closely related to string objects in a variety of other ways.

```
class bytes([source[, encoding[, errors]]])
```

Firstly, the syntax for bytes literals is largely the same as that for string literals, except that a `b` prefix is added:

- Single quotes: `b'still allows embedded "double" quotes'`
- Double quotes: `b"still allows embedded 'single' quotes".`
- Triple quoted: `b'''3 single quotes'''`, `b"""3 double quotes"""`

Only ASCII characters are permitted in bytes literals (regardless of the declared source code encoding). Any binary values over 127 must be entered into bytes literals using the appropriate escape sequence.

As with string literals, bytes literals may also use a `r` prefix to disable processing of escape sequences. See

Arrays of bytes?

Bytearray objects—Python documentation

Bytearray Objects

`bytearray` objects are a mutable counterpart to `bytes` objects.

`class bytearray([source[, encoding[, errors]])`

There is no dedicated literal syntax for bytearray objects, instead they are always created by calling the constructor:

- Creating an empty instance: `bytearray()`
- Creating a zero-filled instance with a given length: `bytearray(10)`
- From an iterable of integers: `bytearray(range(20))`
- Copying existing binary data via the buffer protocol: `bytearray(b'Hi!')`

As bytearray objects are mutable, they support the `mutable` sequence operations in addition to the common bytes and bytearray operations described in [Bytes and Bytearray Operations](#).

Also see the `bytearray` built-in.

Since 2 hexadecimal digits correspond precisely to a single byte, hexadecimal numbers are a commonly used format for describing binary data. Accordingly, the bytearray type has an additional class method to read data in that format:

`classmethod fromhex(string)`

This `bytearray` class method returns bytearray object, decoding the given string object. The string must

Dictionaries

Mapping types--Python documentation

```
class dict(**kwarg)
class dict(mapping, **kwarg)
class dict(iterable, **kwarg)
```

Return a new dictionary initialized from an optional positional argument and a possibly empty set of keyword arguments.

Dictionaries can be created by several means:

- Use a comma-separated list of key: value pairs within braces: {'jack': 4098, 'sjoerd': 4127} or {4098: 'jack', 4127: 'sjoerd'}
- Use a dict comprehension: {}, {x: x ** 2 for x in range(10)}
- Use the type constructor: dict(), dict([('foo', 100), ('bar', 200)]), dict(foo=100, bar=200)

Seriously, the documentation has everything you need!

These are the operations that dictionaries support (and therefore, custom mapping types should support too):

list(d)

Return a list of all the keys used in the dictionary *d*.

len(d)

Return the number of items in the dictionary *d*.

d[key]

Return the item of *d* with key *key*. Raises a `KeyError` if *key* is not in the map.

If a subclass of dict defines a method `__missing__()` and *key* is not present, the *d*[*key*] operation calls that method with the key *key* as argument. The *d*[*key*] operation then returns or raises whatever is returned or raised by the `__missing__(key)` call. No other operations or methods invoke `__missing__()`. If `__missing__()` is not defined, `KeyError` is raised. `__missing__()` must be a method; it cannot be an instance variable:

```
>>> class Counter(dict):
...     def __missing__(self, key):
...         return 0
>>> c = Counter()
>>> c['red']
0
>>> c['red'] += 1
>>> c['red']
1
```

The example above shows part of the implementation of `collections.Counter`. A different `__missing__` method is used by `collections.defaultdict`.

d[key] = value

Final thoughts

- Other Python resources we recommend:
<https://docs.python.org/3/tutorial/>
<https://www.geeksforgeeks.org/python-programming-language/>
<https://pythontutor.com/visualize.html>
- When in doubt: Google for official documentation
- Use VS Code! Really helpful when learning syntax and debugging code.



Lab 1 Assignment

Generating the repository

Note: this is the same process you'll follow to generate the starter code for projects!

Make sure you are signed into your Github account

Set your repository as **Private**

Click **Create repository from template**

After a few seconds, you should see your repository with the starter code

Create a new repository from lab1

The new repository will start with the same files and folders as 388f22/lab1.

Owner * Repository name *

yottalogical / eeecs388-lab1

Great repository names are short and memorable. Need inspiration? How about shiny-spork?

Description (optional)

☐ Public
Anyone on the Internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

☐ Include all branches
Copy all branches from 388f22/lab1 and not just main.

📌 You are creating a private repository in your personal account.

Create repository from template

yottalogical / eeecs388-lab1 Private

generated from 388f22/lab1

<> Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

yottalogical Initial commit 5f5fe4a now 1 commit

.devcontainer	Initial commit	now
.vscode	Initial commit	now
openssl_output.txt	Initial commit	now
part1.py	Initial commit	now
part2.py	Initial commit	now
part3.py	Initial commit	now
part4.py	Initial commit	now
unickname.txt	Initial commit	now

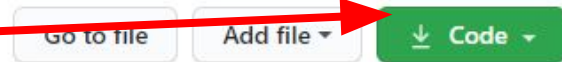
Step 1: let's git it

- Open a terminal
- Verify that git is installed (`git --version`)
- Initial git setup:

```
git config --global user.name "github_id"  
git config --global user.email "github_email"
```

- Clone your repo: find the full URL here!

```
git clone <...>.git  
cd eecs388-lab1
```



7645181 on Sep 4, 2020 1 commit

4 months ago



Push it!

```
$ git add .           # Add all files to your local staging area
$ git commit -m "Pushing some code"  # Create a commit (like a checkpoint!)
$ git push            # Send commit to remote branch
```





See you next week!

Exercise 2: Strings, Lists, and Loops

Let's write a program that...

1. Initializes a list of names, called *students*, with the names of 3 people sitting around you :)
2. Sorts the list in alphabetical order and prints it
3. Saves the first name in a variable *first_name*
4. Deletes the last character from *first_name*, and prints *first_name*
5. Loops through *students*, finds the longest name, and prints it



Exercise 2: Strings, Lists, and Loops

One possible solution...

```
students = ["Jill", "Jacqueline", "Marcus"]
students.sort()

print(students)

first_name = students[0]
first_name = first_name[:-1]
# [-1] gives us the last character of the string
# [:-1] means "give me everything UP TO the last character"
print(first_name)

longest_name = ''
for student in students:
    if len(student) > len(longest_name):
        longest_name = student
print(longest_name)
```

