Website Design

CSc 47300

By the way, I’m Donato Cruz. Nice to meet you (･_･)ノ

Week 1

What Class Is This Again?

Just to make sure you’re in the right place:

- Course Name: **Website Design**
- Course Number: **CSc 47300**

What’s it About?

- It’s about making websites (er… web apps?)
- Identifying common web applications performance pitfalls
- Developing strategies to prevent or mitigate common performance pitfalls

Overview

Full Stack Web Development

This course is a practical, hands-on introduction to creating modern(ish) web applications.

We'll cover (roughly in this order):

- Python
- Server Side Programming
- Client Side Programming
- Trendy Client Frameworks / Libraries
- Development Tools
- Recognizing and Addressing Common Web Application Pain Points

Python Topics

- Basics (Types, Operators, Control Structures, etc.)
- Object Oriented Programming (Classes, Objects)
- Running (and Maybe Writing) Tests

Server Side Programming Topics

- Server Side Language - Python
- Protocol - HTTP/S
- Web Framework - Flask
- Authentication and Session Management - JWT
- Storing Data and Using a Database Abstraction Layer - PostgreSQL and SQLAlchemy respectively
- Building and Consuming APIs
- Deployment Strategies

Client Side Topics

- Fundamentals - CSS/DOM/HTML5
- DOM Manipulation - JavaScript (ES6)
- Asynchronous Request - Fetch API
- Libraries - React
- Framework - Next.js

Development Tools Topic

- Version Control - Git
- Linter - PyLint, ESLint
- Unit Testing Tools - PyTest, Jest
- (Optionally) Debugger - Inspector, Chrome Developer Tools

Motivation for Content

- Practical Coursework
- Freedom To Create
- Technical Interviews
    - Take-home Projects
    - System Design Interviews

Motivation for Technologies

- Highly applicable across various career paths (Data Engineer, Data Scientist, Software Engineer)
- Straightforward to install entire stack (Python, PostgreSQL) on Windows, Linux, and OSX
- Develop a skill set that's currently sought after
- A fun and easy going stack (really!)

Me?

- Have worked professionally as a Software Engineer for over 8 years
    - Technologies used:
        - Golang (gRPC), Python (Django, Flask), Ruby (Rails, Sinatra), JavaScript (React)
- Have taught for over 8 years
    - CUNY Tech Prep
        - CUNY Tech Prep is a year long technical and professional development program for CUNY computer science majors to learn in-demand technologies, master professional soft skills, and land great tech jobs in NYC.
    - Tech-in-Residence Corps
        - The Tech-in-Residence Corps brings industry professionals from the NYC tech ecosystem into CUNY classrooms to transfer their skill sets and applied knowledge directly to students and to collaborate with Computer Science faculty.

You?

- Who You Be?
  - Name, Year, Fun Fact
- What are you hoping to get out of this course?

About... You

I expect (hope?) that you:

- Are very comfortable using the command line
- Have the ability to install tools, software, etc. … and troubleshoot installations
- Are able to navigate through your file system (both through a file explorer like Finder and through the command line)
- Have basic/rudimentary knowledge of HTML and CSS

Also, a bib (non-) expectation:

- Minimal experience with Python, JavaScript, server-side web development, and modern front-end development

Workload

- Project Proposal & Presentation
  - The proposed project will essentially be a single web application based on the material we've learned
  - Two touchpoints: the first is the middle of the semester, and the second is the end of the semester.
    - Each touchpoint is an opportunity for you to demonstrate what you've built
  - The proposed project is the deliverable for the course
- 3-5 Assignments
  - Write your own code!
  - Some examples:
    - (Server Side) Create an API that serves Restaurant Inspection Results
    - (Client Side) Create a React App that provides a visual interface to Restaurant Inspection Results

Difficulty Level

This course is not challenging in the way that something like algorithms is, but it’s challenging because of:

- The wide range of topics covered
- The difficult nature of debugging web applications that involve integrating several technologies

About that Homework

On the subject of homework and difficulty level, if you need help:

- Please ask on forums (Slack)
  - Please do not post significant parts of the homework solution
- High level discussions amongst yourselves is fine
- Help debugging an exception/error amongst yourselves is fine
- See me (office hours?)

Using online resources outside of the course materials…

- Is ok if it’s just a line or two and you annotate your code with a comment and a link
- Is not ok if you’re lifting a significant amount of code from tutorials or projects found online

Writing Your Own Code

Whatever you do, though… write your own code!

- Don’t copy anyone else’s code
- Don’t distribute/publish your code (including publishing to a public git repository)
    - You can publish once the class is over

If You Got Anything Out of These Slides

- You’re going to be writing a bunch of JavaScript and Python
- I’m available for help
    - Best way to get in touch with me is via email, Slack, or office hours.
- Write your own code!

Syllabus Review

Any Questions?

Small Asks

Join Slack!

Join Github!

Take a Breather!

Introduction to Python

A Little Bit About Python

What are we getting ourselves into? We'll introduce Python by:

1. Defining what Python is
2. Taking a quick look at some interesting Python features
3. Going over why we're using Python
4. And, finally, we'll discuss how to run Python programs

About Python

- High-level programming language available on many platforms
- Strongly, Dynamically Typed
- Supports multiple programming paradigms
  - Structured (Procedural), Object-Oriented, and Functional Programming
- Interpreted Language (i.e., not compiled)
- General-Purpose Language
  - Data Engineering
  - Data Science
  - Software Engineering
- Easy to use

Python Constructs

Things we will take a look at:

- Python’s types
- Numbers and numeric operators
- Strings and string operators
- Booleans and logical and comparison operators

Some Definitions

- Value: Data
  - name = “Donato”
- Type: A category or classification of values
  - type(1) -> <type 'int'>
- Operator: A language construct that allows the manipulation or combination of a value or values to yield another value
  - +, -, /, *
- Operand: A value that an operator works on; the subject of an operator
  - 5 + 10
- Unary Operator: An operator that only has one operand.
  - -100

Some Definitions (cont.)

- Binary Operator: An operator that has two operands
  - 100 - 50
- Prefix Operator: An operator that goes before (to the left) of its operand(s)
  - age = 100; ++age
- Infix Operator: An operator that goes between its operands
  - age = 100; age++

Let's start off with Comments

Single-line and multi-line comments are one in the same 😔

```
# mood = "excited"
```

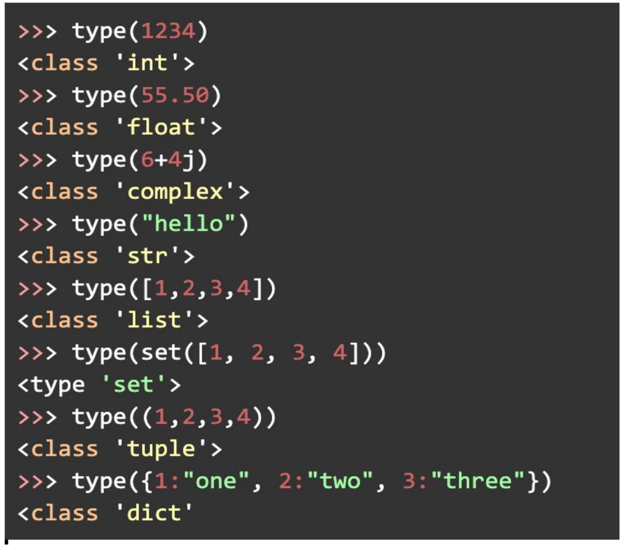
```
# day_of_week = "tuesday"
# mood = "happy"
# print(f"Hi, today is {day_of_week} and I am feeling {mood}.")
```

Values

What's a value?

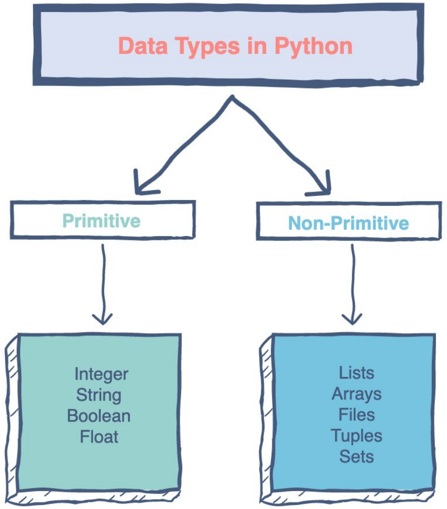
- Values are just data.
- They're pieces of information.

Some examples of literal values in Python:



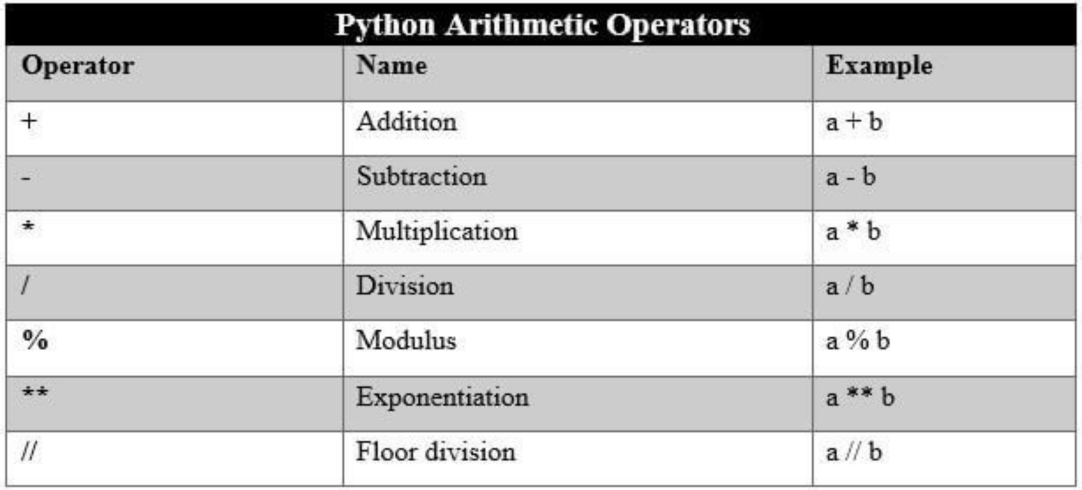
```
>>> type(1234)
<class 'int'>
>>> type(55.50)
<class 'float'>
>>> type(6+4j)
<class 'complex'>
>>> type("hello")
<class 'str'>
>>> type([1,2,3,4])
<class 'list'>
>>> type(set([1, 2, 3, 4]))
<type 'set'>
>>> type((1,2,3,4))
<class 'tuple'>
>>> type({1:"one", 2:"two", 3:"three"})
<class 'dict'
```

Primitive vs. Non-Primitive Types



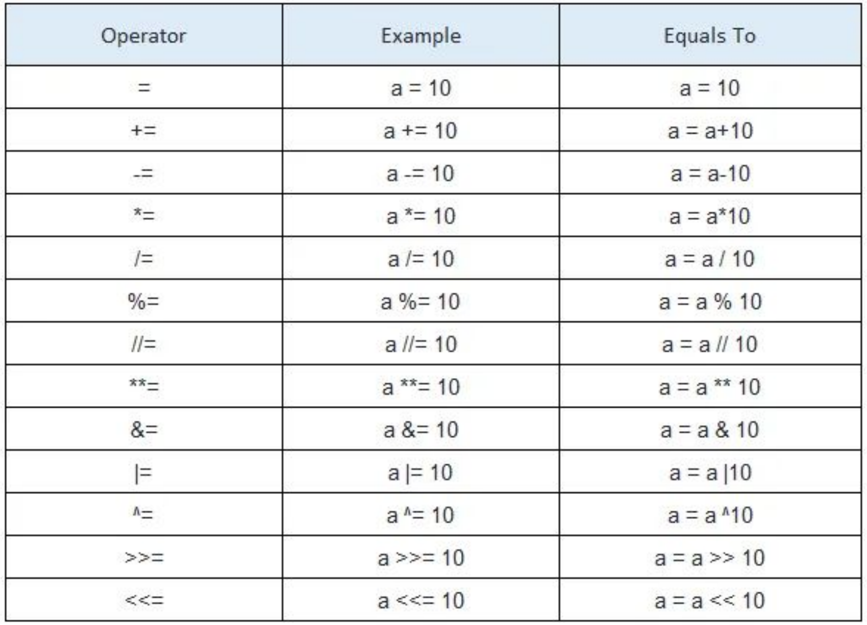
- Primitive data types are the building blocks for data manipulation and contain pure, simple values of data.
- Non-primitive data types are the sophisticated members of the data types family. They don't just store a value, but rather a collection of values in various formats.

Operators



Python Arithmetic Operators		
Operator	**Name**	**Example**
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b
**	Exponentiation	a ** b
//	Floor division	a // b

Operators (cont.)



Operator	Example	Equals To
=	a = 10	a = 10
+=	a += 10	a = a+10
-=	a -= 10	a = a-10
*=	a *= 10	a = a*10
/=	a /= 10	a = a / 10
%=	a %= 10	a = a % 10
//=	a //= 10	a = a // 10
**=	a **= 10	a = a ** 10
&=	a &= 10	a = a & 10
|=	a |= 10	a = a |10
^=	a ^= 10	a = a ^10
>>=	a >>= 10	a = a >> 10
<<=	a <<= 10	a = a << 10

Logical Operators

- not

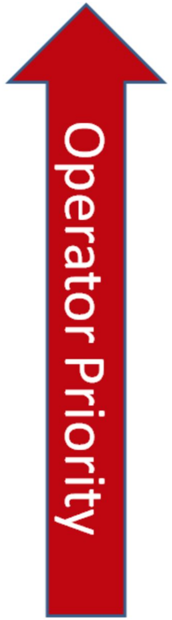
x	not x
False	True
True	False

- and

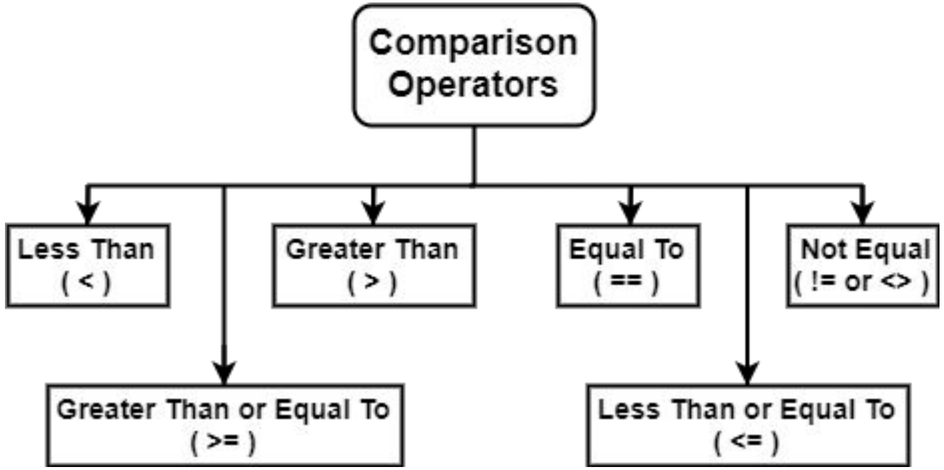
x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- or

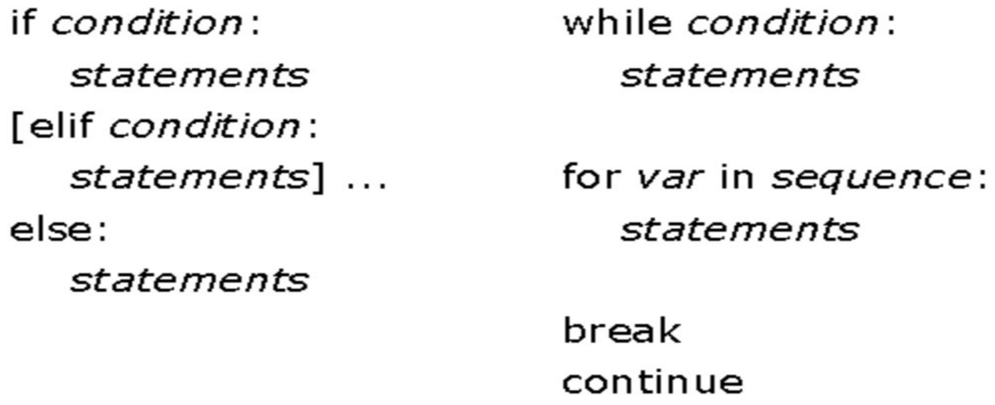
x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True



Comparison Operators



Control Structures



```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements
```

```
while condition:
    statements

for var in sequence:
    statements

break
continue
```

A few Notes on How You'll be Writing Programs

- Use any text editor you want
- Some editors that I, as well as my colleagues, have used in the past:
    - Emacs
    - PyCharm (for a full IDE experience)
    - Sublime Text
    - Vim
    - VSCode

Choose the one that you're most comfortable with!

Some Quick Exercise

A silly practice program to get you warmed up

FizzBuzz

FizzBuzz

- write a program that uses print to print all the numbers from 1 to 100
- for numbers divisible by 3, print "Fizz" instead of the number
- for numbers divisible by 5 (and not 3), print "Buzz" instead
- for numbers divisible by both 3 and 5, print "FizzBuzz" instead

Functions

Back to Definitions

- function - a named sequence of statements that performs a specific task or useful operation
- parameter - a variable that receives an argument that is passed into a function, think of it as as the variable(s) in the function header / signature
- call/invoke/apply - to run or execute a function
- argument - a piece of data that is passed into a function when that function is called
- scope - the area of the code where a name/identifier is available for access and/or use

Function Example

```
import requests

def print_pokemon_details(pokemon_name):
  response = requests.get("https://pokeapi.co/api/v2/pokemon/{}".format(pokemon_name))
  data = response.json()

  print("id={}, name={}, height={}, weight={}".format(
    data['id'],
    data['name'],
    data['height'],
    data['weight'],
    )
  )

print_pokemon_details("pikachu")
```

Classes

Back to Definitions

- Classes - provide a means of bundling data and functionality together.
- Objects - Instances of classes, each containing its own state.
- Constructor - Method defined on the class used to instantiate an object.
  - Default constructor
  - Parameterized constructor

Class Example

```python
class Vehicle:
    def __init__(self, color, doors, tires, vehicle_type):
        self.color = color
        self.doors = doors
        self.tires = tires
        self.vehicle_type = vehicle_type

    def brake(self):
        """
        Stop the car
        """
        return "{} braking".format(self.vehicle_type)

    def drive(self):
        """
        Drive the car
        """
        return "I'm driving a {} {}!".format(self.color, self.vehicle_type)

if __name__ == "__main__":
    car = Vehicle("blue", 5, 4, "car")
    print(car.brake())
    print(car.drive())

    truck = Vehicle("red", 3, 6, "truck")
    print(truck.drive())
    print(truck.brake())
```

Supplemental Resources

- [Free] Python 101
  - Gain familiarity with the language constructs through intuitive examples.
- [Paid] Learn Python The Hard Way
  - A bit more thorough and example driven learning.

PyEnv

PyEnv

PyEnv → Simple Python Version Management

PyEnv lets you easily:

- Switch between multiple versions of Python
- Provide support for per-project Python versions

PyEnv can be installed via your respective Package Manager

- Homebrew (macOS)
    - `brew install pyenv`

PIP

PIP

PIP is Python’s official Package manager.

Does anyone know of any other package managers for other languages?

PIP

PIP is Python’s official Package manager.

Does anyone know of any other package managers for other languages?

- gem for Ruby

PIP

PIP is Python's official Package manager.

Does anyone know of any other package managers for other languages?

- gem for Ruby
- npm for Node

PIP

PIP is Python’s official Package manager.

Does anyone know of any other package managers for other languages?

- gem for Ruby
- npm for Node
- composer for PHP

PIP

PIP is Python’s official Package manager.

Does anyone know of any other package managers for other languages?

- gem for Ruby
- npm for Node
- composer for PHP

Among other things, pip allows you to download and install libraries, as well as remove and upgrade them.

PIP (cont.)

Unfortunately, by default PIP install packages globally (system-wide).

However, we can circumvent that limitation by leveraging virtualenv, which allow us to create isolated environments that not susceptible to conflicts in shared libraries dependencies.

VirtualEnv

Virtualenv

Virtualenv → a tool to create isolated Python environments, which allows you to work on a specific project without worry of affecting other projects.

PyEnv provides virtualenv as a shim.

Local vs Global Installation

You can install libraries globally if you need to (and it will be probably necessary for some things) by installing the library outside of a virtual environment.

- Some libraries are actually command line tools that you want to use throughout your system.
  - Install these globally.
- While other libraries are for specific applications that you are developing.
  - Install these locally.

More About Global Package Installation

Why do we want to avoid installing libraries globally?

More About Global Package Installation

Why do we want to avoid installing libraries globally?

- Multiple applications, different dependencies

More About Global Package Installation

Why do we want to avoid installing libraries globally?

- Multiple applications, different dependencies
- Perhaps even OS level dependencies

Requirements.txt

Lastly, PIP can use a file called requirements.txt to store dependencies.

This this usually be placed in the root of your project folder.

If your program depends on a set of libraries

- It may be a good idea to put that library in requirements.txt
- … so you don’t have to remember all of the requirements, and they can be installed all at once.
- Also, listing your dependencies makes it easy for others to clone and run your project with minimal friction.

Libraries

So… what is PIP installing? What are these libraries anyway?

Libraries are just Python files.

- You can bring in the code from one file into another using the imports keyword.

Core Libraries

Some libraries are available without having to create or download them.

A couple of useful core libraries include:

- Math - Mathematical Functions
- Time - Time access and conversions

Downloaded Libraries

Of course, we're not stuck with just using the core libraries. We could download pre-built libraries as well.

Downloaded Libraries (cont.)

How to install a library:

```
pip install requests
```

How to use a library:

```
import requests

# https://pokeapi.co/docs/v2#pokemon
response = requests.get("https://pokeapi.co/api/v2/pokemon/charmander")
data = response.json()

print("id={}, name={}, height={}, weight={}".format(
    data['id'],
    data['name'],
    data['height'],
    data['weight'],
    )
  )
```

All Together

- Install PyEnv
  - `brew install pyenv`
- Install Python 3
  - `pyenv install 3.10.6`
- Install PyEnv-VirtualEnv
  - `brew install pyenv-virtualenv`
- Create a Virtual Environment
  - `pyenv virtualenv venv`
- Activate The Virtual Environment
  - `pyenv activate venv`
- Install Listed Dependencies
  - `pip install -r requirements.txt`
- Execute Demo
  - `python http_request.py`

Git

We’re Using Git!

- Git is the version control system that we’re using
- It’s a modern distributed version control system
- It has emerged as the standard version control system to use
- (Some others are…)
    - Mercurial
    - Subversion (SVN)

A Bit About Git

It was developed by Linus Torvalds… who?

(The guy who made Linux)

Github vs Git

Github and Git are different!

- Github is a website that hosts git repositories
- On its own, Git is just version control

Who Uses Git and Github?

Git is used to maintain a variety of projects, like:

- Twitter's Bootstrap
- Ruby on Rails

We’re Using Git, a Distributed Version Control System

Some Terminology

- Repository - the place where your version control system stores the snapshots that you save
  - Think of it as the place where you store all previous/saved versions of your files
  - This could be:
    - Local - on your computer
    - Remote - a copy of versions of your files on another computer
- Git - The distributed version control system that we're using
- Github - A website that can serve as a remote repository for your project

What's a remote repository again?

- A copy of versions of your files on another computer/server

Where Are My Files

In your local repository, git stores your files and versions of your files in a few different conceptual places:

- The working directory / working copy - stores the version of the files that you're currently modifying / working on
- Index - the staging area where you put stuff that you want to save (or… that you're about to commit)
- HEAD - the most recent saved version of your files (or… the last commit that you made)

Aaaaand. More Terminology.

- Commit → save a snapshot of your work
- Diff → the line-by-line difference between two files or sets of files

Two Basic Workflows

1. Creating and setting up local and remote repositories.
2. Making, saving, and sharing changes.

Creating Repositories

- Create a local repository
- Configure it to use your name and email (for tracking purposes)
- Create a remote repository
- Link the two

Making, Saving, and Sharing Changes

- Make changes
- Put them aside so that they can be staged for saving / committting
- Save / Commit
- Send changes from local repository to remote repository

Ok. Great!

Remind me again, what’s Github?

- Github is a website
- It can serve as a remote Git repository
  - That means it can store all versions of your files
  - (After you’ve sent changes to it)

git clone

Creates a local repository from a remote one

```
git clone REPOSITORY_URL
```

REPOSITORY_URL is usually going to be something that you copy from Github.

git init

Creates a new local repository

- You can tell a repository is created by running `ls -l` ... it creates a `.git` directory

git config

Configure your user name and email for your commits

- This has nothing to do with you computer’s account or your account on Github.
- This information helps track changes.

```
git config user.name  "foo bar baz" ; git config user.email foo@bar.baz
```

git remote add

Add a remote repository so that you can synchronize changes between it and your local repository.

```
git remote add REPOSITORY_NAME REPOSITORY_URL
```

Typical Workflow for Making Changes

1. Make changes
2. `git status`(to see what changes there are)
3. `git add -all`(to stage your changes for committing)
4. `git status`(to see your staged changes)
5. `git commit -m ‘Message goes here.’` (to save your changes)
6. `git push origin master`(optional send/share your changes to a remote repository)

git status

git status → show what changes are ready to be committed as well as changes that you are working on in your working directory that haven't been staged yet

```
git status
```

git add

git add → make a change to be staged

Add all files

- `git add --all`

Add specific file

- `git add file_path/file_name.file_extension`

git commit

git commit → take a snapshot of your work

```
git commit -m ‘Message goes here.’
```

git log

git log → show commit history of your repository or file

```
git log
```

git diff

git diff → show the line-by-line differences between your last commit and your working directory

```
git diff
```

git reset

git reset → revert last commit… or unstage changes

Revert last commit:

- `git reset HEAD^1`

Unstage changes:

- `git reset file_path/file_name.file_extension`

git push

git push → send your code to a remote repository

```
git push
```