4. More Git

And a little Python

Agenda 📅

1. Git Branching Recap
2. Pull Requests
3. Teamwork Tips
4. Python Teaser

Logistics ⚙️

- Groups Assigned! See the assignment!
- Homework 4 is a **group assignment** (due the usual time)!
- There's a **class GitHub Organization** now. You'll need to be in it to turn in coding homework. Let me know if you didn't get an invite!
- Meta is holding **Mock Interviews** in the second week of October. I'll send a link in the Discord.
- Programming Club events:
  - HackJam starts **Monday**. Great way to build more project experience if that's something you feel like you're missing.
  - Mock Interviews this **Sunday** from 2-4pm

Recap: Git

Which of the following is considered harmful when using Git?

1. Committing frequently and in small chunks
2. Creating branches when you work on new features or bugs
3. Pushing directly to the main branch
4. Using GitHub Desktop to manage your local repository state

Recap: Git

Which of the following is considered harmful when using Git?

1. Committing frequently and in small chunks
2. Creating branches when you work on new features or bugs
3. Pushing directly to the main branch
4. Using GitHub Desktop to manage your local repository state

Recap: Git

Which of these word salads will successfully publish your code to GitHub, assuming you have a branch called main and origin points to a GitHub repo:

1. git origin main push
2. git push origin main
3. git push main origin
4. git origin push main

Recap: Git

Which of these word salads will successfully publish your code to GitHub, assuming you have a branch called main and origin points to a GitHub repo:

1. git origin main push
2. git push origin main
3. git push main origin
4. git origin push main

Recap: Git

There are three major git commands related to branching. What are they?

Recap: Git

There are three major git commands related to branching. What are they?

git branch, git checkout, git merge

Pull Requests (Group Project)

Choose your fighter (again)!

Suppose you're going to develop an app as a team using a GitHub repository. Which setup is best?

1. Team members can make code changes to main whenever they want
2. No changes to main can be made without someone else on the team approving them.
3. Most changes can be made to main without approval, but really important ones require review.

Pull Requests (PRs)

- Confusingly named -- it’s really more like a “Push Request”
- PRs are just sets of changes that you’re asking someone else to accept and merge into the working code copy.
- Here, the merge happens **in GitHub**, not **on your local machine**.
- You can open a PR between any two branches, but the most common use case is merging from a feature or development branch into main.
- This is where code review happens!

In the group project

...you will do each task on a **branch**. When that branch is ready to go, you will open a **pull request** to the main branch. Someone else from the team will have to **review** that pull request before it can be merged in.

Workflow (Pull Requests)

1. Get the latest version of main (i.e. your teammates' work): **`git pull origin main`**
2. Create a new branch for a feature: **`git branch {branch_name}`**
3. Switch to new branch: **`git checkout {branch_name}`**
4. Change (create, delete, edit) files (just for that feature).
5. Check what files and changes git is tracking: **`git status`**
6. Prepare files for commit: **`git add {filename1} {filename2}`** or **`git add -A`**
7. Officially save those changes in the project history: **`git commit -m "Description of changes"`**
8. Add your changes to your remote repo: **`git push origin {branch_name}`**
9. Create a Pull Request (PR) from your branch to main.

Pro Tips

- The biggest (normal) PR I've had at work was around ~200 lines. **Anything bigger than 200 lines is probably bad** and needs to be split up!
- All feature development (including addressing bugs) should be on a separate branch. **Don't ever develop on main and push to main.**

It’s demo time! 

Pull Requests

Activity: Pull Requests

Grab a partner, and then do the following:

1) Make a public GitHub repo where both of you can contribute (author of the repo should add the other person as a collaborator).
2) Try to cause a merge conflict between two of your Pull Requests.
3) **First team to finish gets an extra late day each.**

No partner? You can do this activity solo if needed. (But you can’t earn a late day.)

Break


GitHub
Taking a picture of your code
Notch Retweeted
Definitely Not Dan @Def_N... · 2h
Replying to @notch
Read your code aloud and put it on Amazon as an audiobook.
9
15
216

Working in teams effectively

Feedback

- Conflict is really scary so most of us naturally avoid it.
- However, one of the skills we are developing in this class is the ability to give constructive feedback.
    - This includes for me!
- If you feel like a group member isn't pulling their weight and you come to me, the first thing I'm going to ask is this: "Have you talked to them about it?"
- (Conversely – don't hold back on telling your teammates that they're crushing it if you feel that way! Showing gratitude is a great way to improve everyone's happiness in a work environment!)
- You'll also be expected to leave peer feedback as part of some group assignments later on.

Team Charter

There are some big questions you want to think about up front:

- How will you make sure work is distributed equitably?
- How will you communicate with each other?
- How will you handle different expectations around work quality and timing?
- **All of this stuff is hard to predict up front! This discussion doesn’t just happen one time, but try to imagine worst-case scenarios.**

Writing feedback: High quality **positive** feedback

High quality positive feedback is **patternistic**, **specific**, and **behavioral**.

- **Patternistic** - Make sure your feedback is about tendencies that people do often, not just one off things that went well!
- **Specific** - "Demitri is a nice person and very pleasant to work with" isn't particularly helpful
  - "Demitri's proactive attitude in helping people whenever he has free time" is a lot more helpful!
- **Behavioral** - Instead of just focusing on people's impact, focus on why they were able to accomplish those things.
  - "Shonda is the most productive engineer on our team and led four projects to completion this half! The reason she's been so productive is that she plans work very meticulously, elicits requirements, and designs her unit tests and CI/CD before she starts coding."

Writing feedback: High quality **constructive** feedback

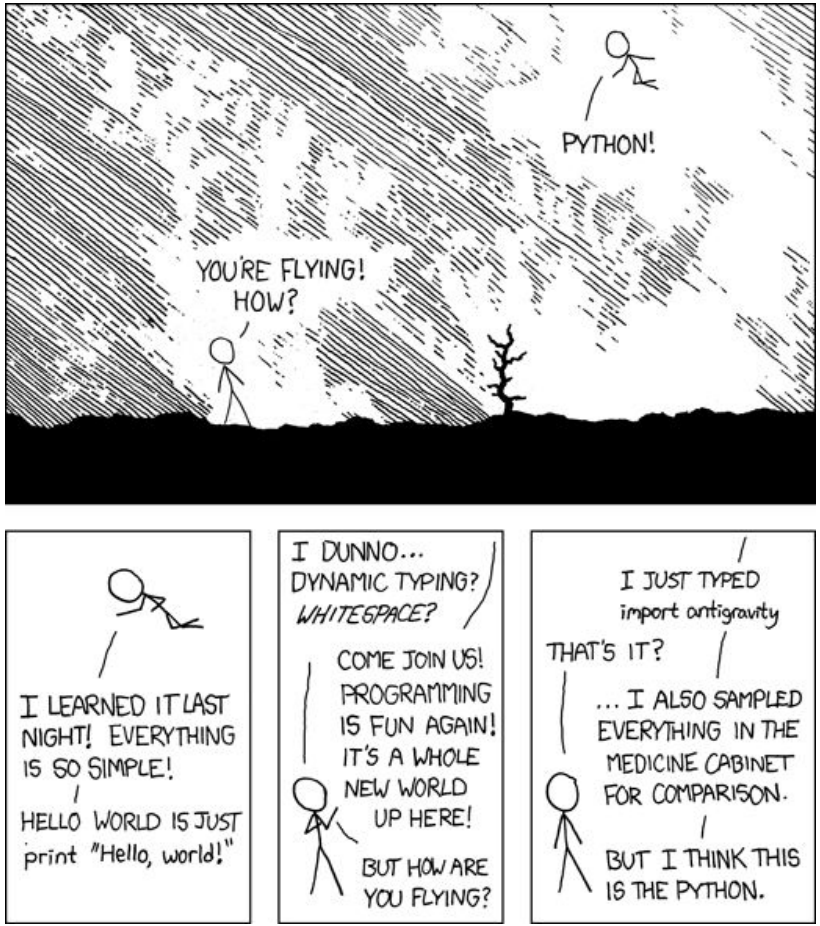
High quality feedback is **actionable**, **contextual**, and **constructive**.

- **Constructive** - Offer feedback only if people can easily change it!
  - "Your teeth are too big for your mouth" vs "You have something in your teeth"
- **Contextual** - Give feedback that demonstrates that you understand **why** the decisions were made the way they were (this is what I'm worst at!).
  - "You didn't participate in Secret Santa, so you clearly don't like us" vs "We missed you during Secret Santa this year! I get it though, layoffs are hard financially, especially during COVID".
- **Actionable** - Suggest specific steps they can follow to improve.
  - "Dinner sucked today" vs "I didn't enjoy dinner today - next time, perhaps we can try using basil instead of garlic."

Writing Feedback: What to **avoid**

- **Don't sandwich constructive feedback.**
  - "Quantum is a wonderful engineer who has had tons of impact. There have been times, though, where Quantum has made inappropriate comments at work. For example, he made a comment about how awful our partner teams are to work with. Other than those instances, I still enjoy working with Quantum on a daily basis!"
  - This diminishes the true importance of your feedback. Quantum needs to address this feedback, so you shouldn't try to hide it!
- **Stick to facts, not intent or feelings.**
  - "Gloria clearly dislikes me - she doesn't review my diffs but reviews other people's."
  - Reframe this to "Of the 73 diffs that Gloria has reviewed in the last four months, only 3 have been my diffs. This has been problematic because I rely on speedy reviews to continue having impact. I want to understand why Gloria has not been able to review my diffs very often."

Python 🐍


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

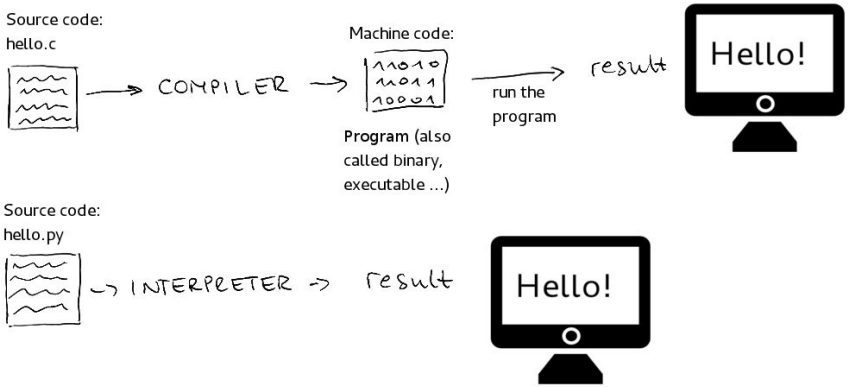
Python is EASIER to write and HARDER to run.

(Another way of saying this: Python is easier on developers and harder on users).

(Your mileage may vary though)

Python vs Java: Big Ideas

- Interpreted vs. Compiled
  - Python is interpreted, Java is compiled
  - Trade-offs of speed & flexibility
- Static vs. Dynamic Typing
  - Java is statically typed, Python is dynamically typed. That basically means that Java requires you to be very explicit about the types of variables, and Python doesn't.
  - Dynamic typing seems pretty sweet but it's tricky in large codebases. It's harder to read someone else's code when you don't know the type of all variables.



Java

```
int x = 0;
```

Python

```
x = 0
```

Python vs Java: Syntax

```java
class Solution
{
    public int fib(int N)
    {
        if(N <= 1)
            return N;

        int a = 0, b = 1;

        while(N-- > 1)
        {
            int sum = a + b;
            a = b;
            b = sum;
        }
        return b;
    }
}
```

Python vs Java: Syntax

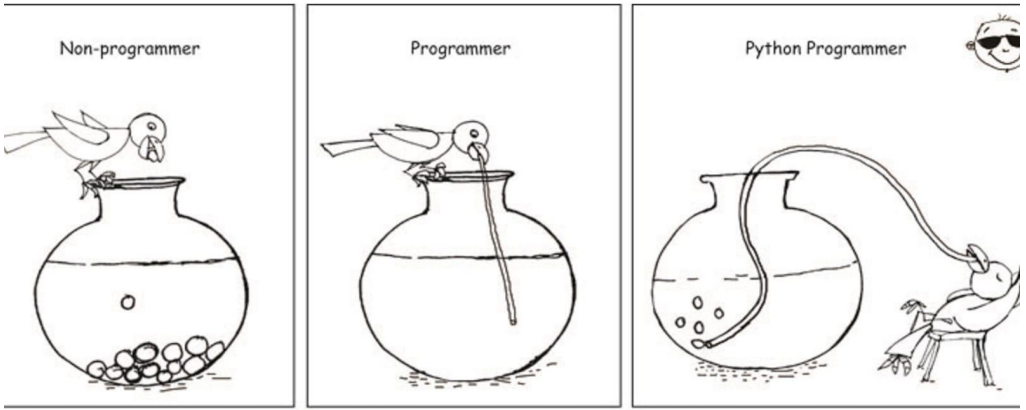
```java
class Solution
{
    public int fib(int N)
    {
        if(N <= 1)
            return N;

        int a = 0, b = 1;

        while(N-- > 1)
        {
            int sum = a + b;
            a = b;
            b = sum;
        }
        return b;
    }
}
```

```python
class Solution:
    def fib(N):
        if N <= 1:
            return N

        a = 0
        b = 1  # could also do a = 0; b = 1

        while N > 1:
            total = a + b
            a = b
            b = total
            N = N - 1  # could also do N -= 1

        return b
```

Python vs Java: Syntax

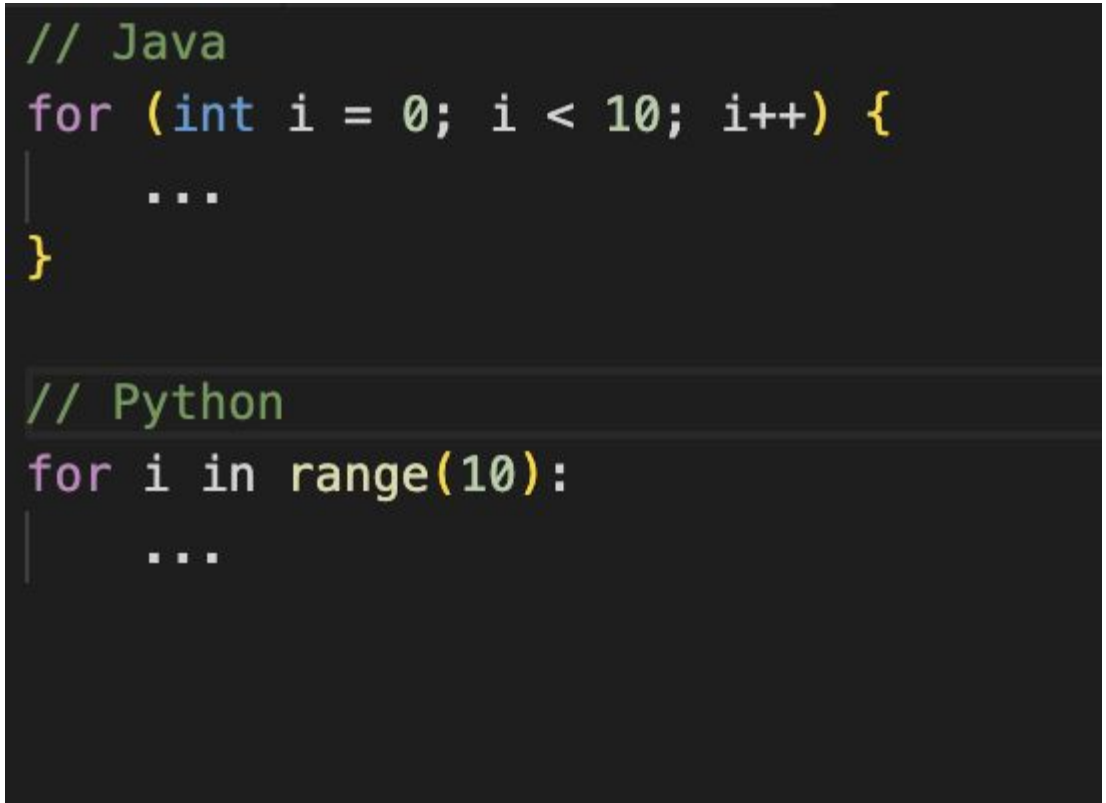
- Colons + indents instead of curly braces
- Use “def” to start functions, “#” to start comments
- “++” and “--” not supported, better to be explicit
  - Stylistically, Python strives to be **readable.**.
- Different syntax for common data structures

```
1  my_set = set()
2  my_set.add("foo")
3  "foo" in my_set  # True
4
5  my_dict = {}
6  my_dict["key"] = "value"
7  my_dict["key"]  # "value"
8
9  my_list = []
10 my_list.append(1)
11 my_list[0]  # 1
12 my_list[0] = 2
13 my_list  # [2]
```



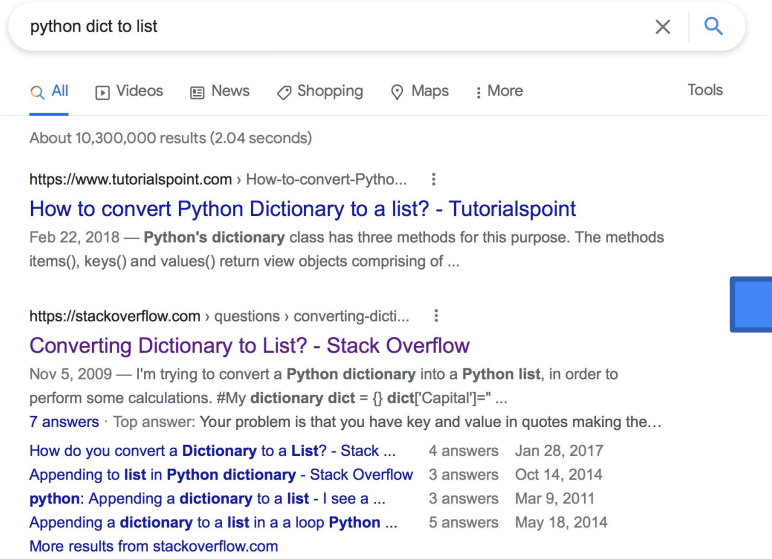
Python vs. Java: For Loops

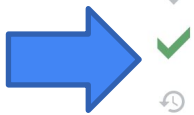
- Python for loops look a little different
- They can only be written as “**for** *variable* **in** *iterable*”
- This means we need constructs like “range()”



Got more Python questions?

- Documentation: https://docs.python.org/3/
- StackOverflow for common snippets:





Your problem is that you have `key` and `value` in quotes making them strings, i.e. you're setting `aKey` to contain the string `"key"` and not the value of the variable `key`. Also, you're not clearing out the `temp` list, so you're adding to it each time, instead of just having two items in it.

To fix your code, try something like:

```
for key, value in dict.iteritems():
    temp = [key,value]
    dictlist.append(temp)
```

You don't need to copy the loop variables `key` and `value` into another variable before using them so I dropped them out. Similarly, you don't need to use append to build up a list, you can just specify it between square brackets as shown above. And we could have done `dictlist.append([key,value])` if we wanted to be as brief as possible.

Or just use `dict.items()` as has been suggested.

Later 👋

- Next class, we finally get to write code!
- Before next class, get together with your group and complete HW4: Team Charter + Pull Requests!