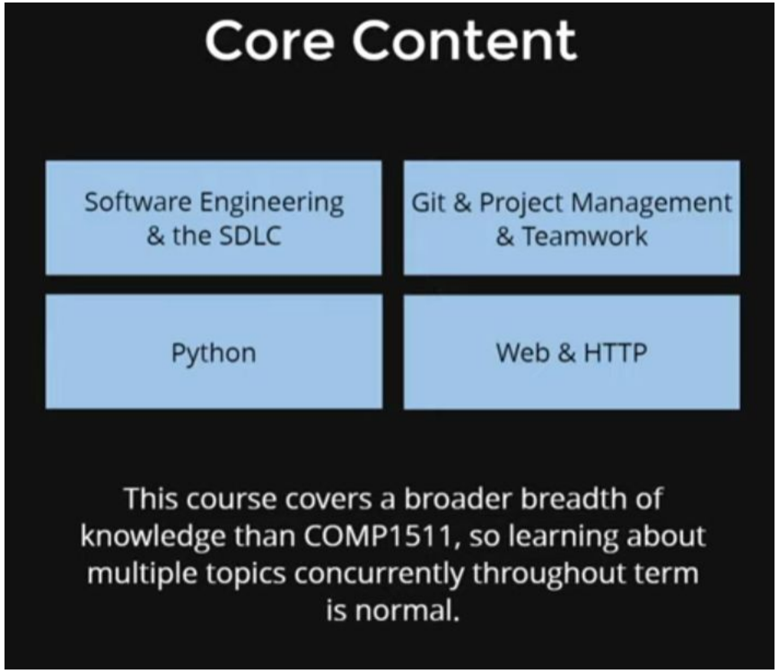
Week 02

No one reads this, bonus participation marks to the first person who emails me to claim the bonus marks

Week 1 Overview


Core Content

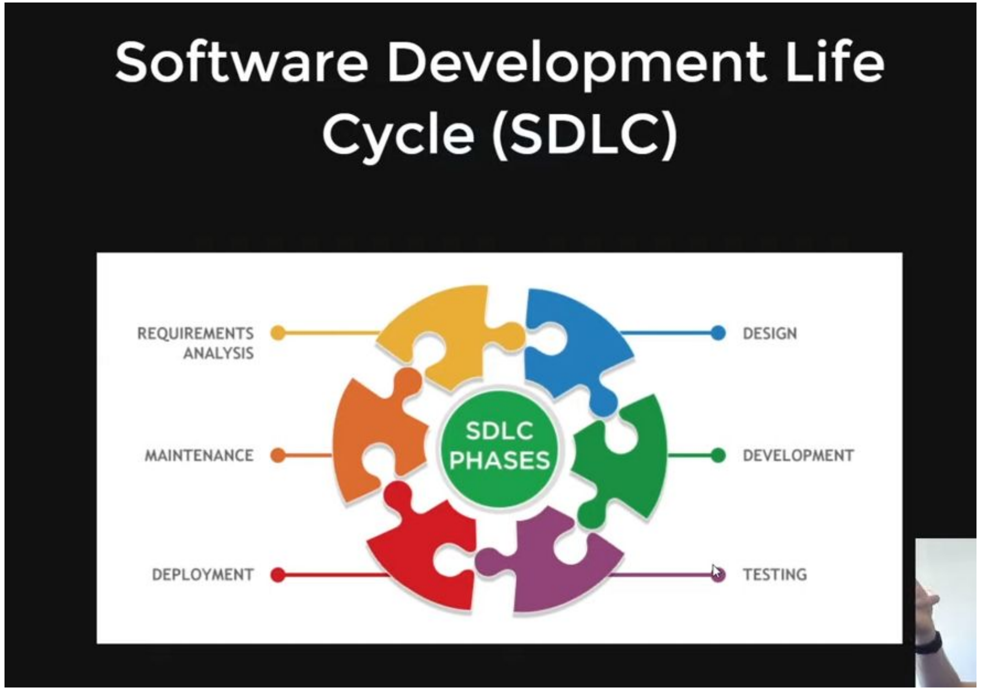
Software Engineering & the SDLC

Git & Project Management & Teamwork

Python

Web & HTTP

This course covers a broader breadth of knowledge than COMP1511, so learning about multiple topics concurrently throughout term is normal.


Software Development Life Cycle (SDLC)

- Good software engineering leads to derisk - Save time, money, doesn't cause issues
- You want software to work, for it to be cheap easy to maintain and grow and edit

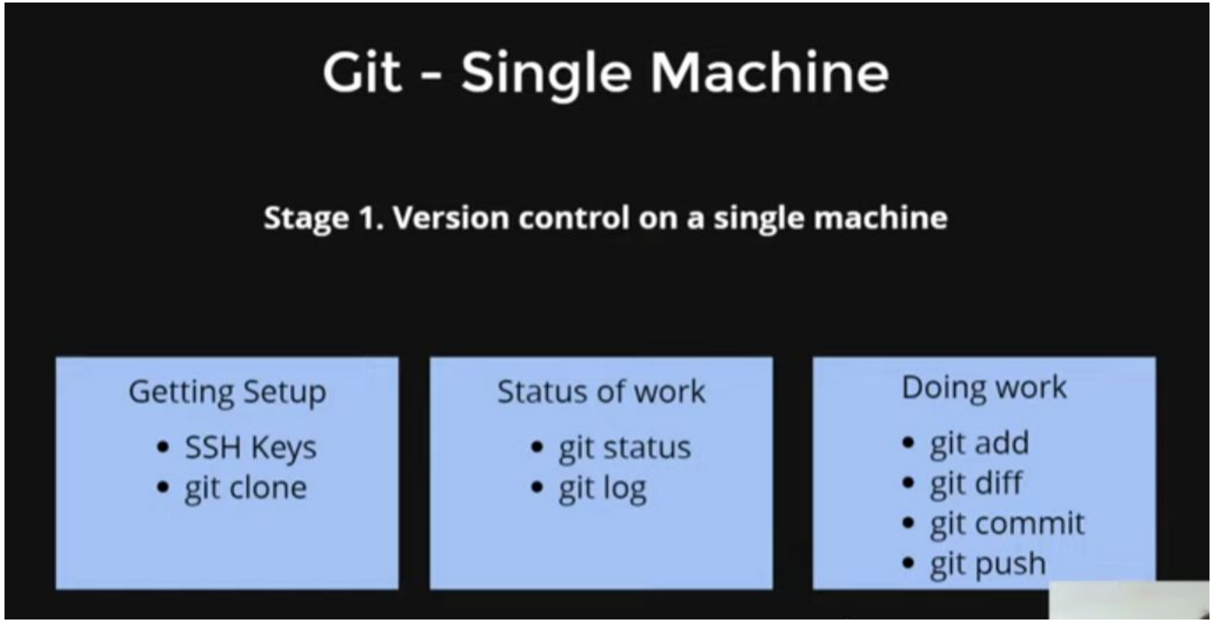
Week 1 Overview - Git

Git is the tool, like a language

Websites like github, gitlab, bitbucket use git

I also use git branch bc I always forget which branch I'm on

Most used: add, commit, push, pull, status, branch



Git - Single Machine

Stage 1. Version control on a single machine

Getting Setup
- SSH Keys
- git clone

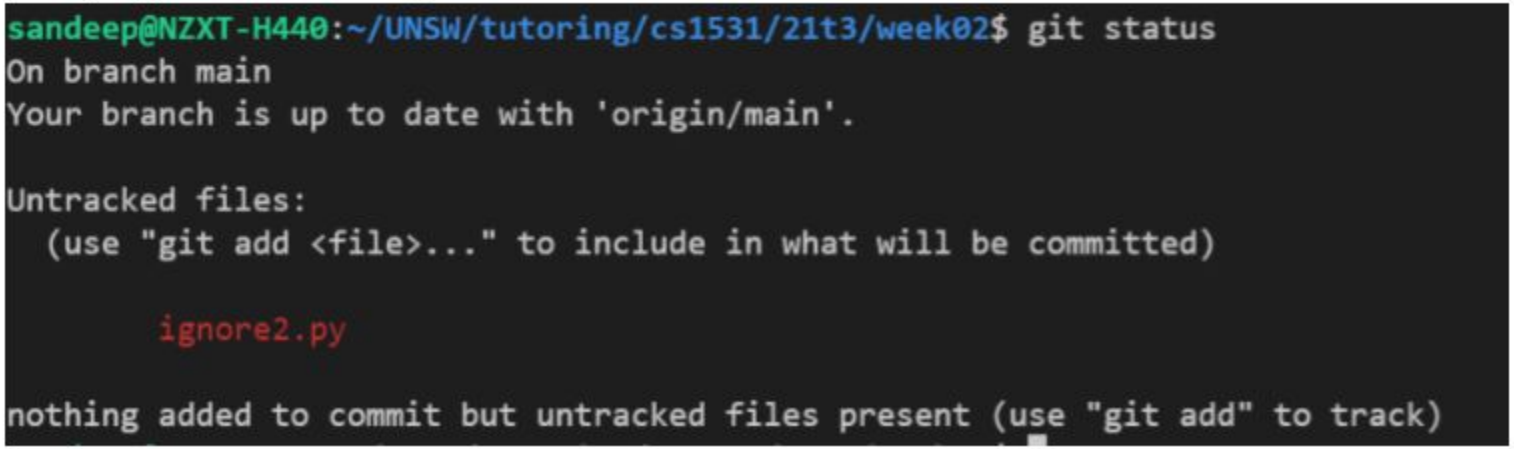
Status of work
- git status
- git log

Doing work
- git add
- git diff
- git commit
- git push

Before git add

- Adding (Staging): Get them ready
- Committing: Snapshot in time

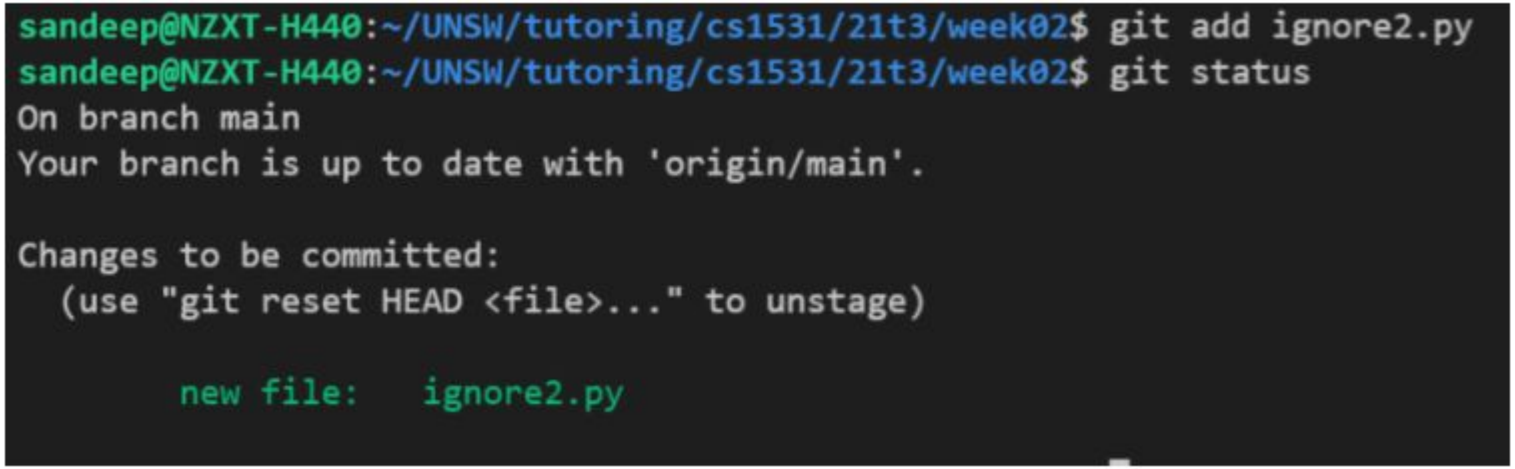
Modifying a file BEFORE git add



```
sandeep@NZXT-H440:~/UNSW/tutoring/cs1531/21t3/week02$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ignore2.py

nothing added to commit but untracked files present (use "git add" to track)
```

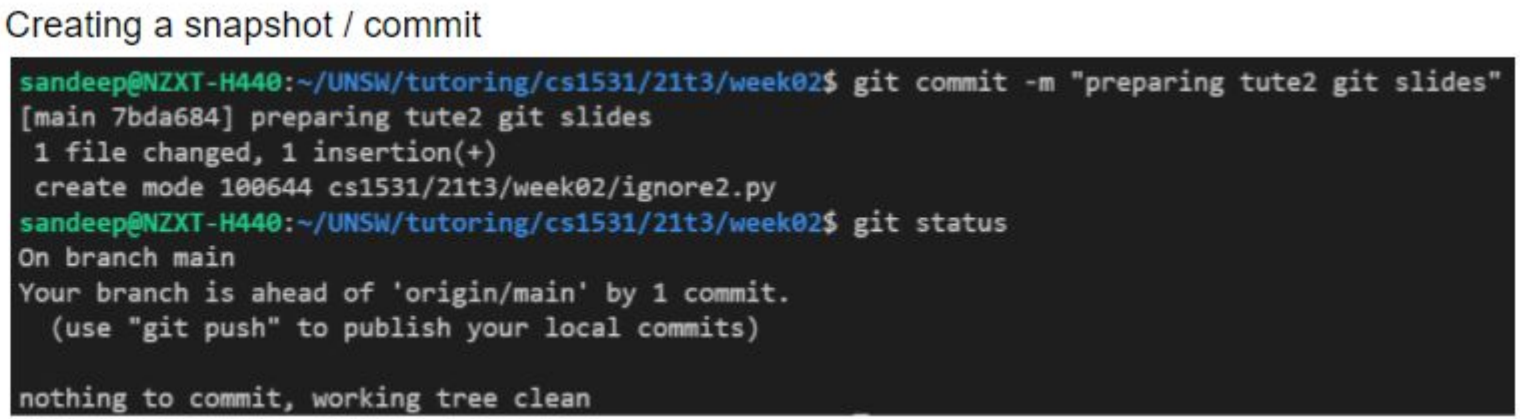
After git add

Git add gets the files ready:



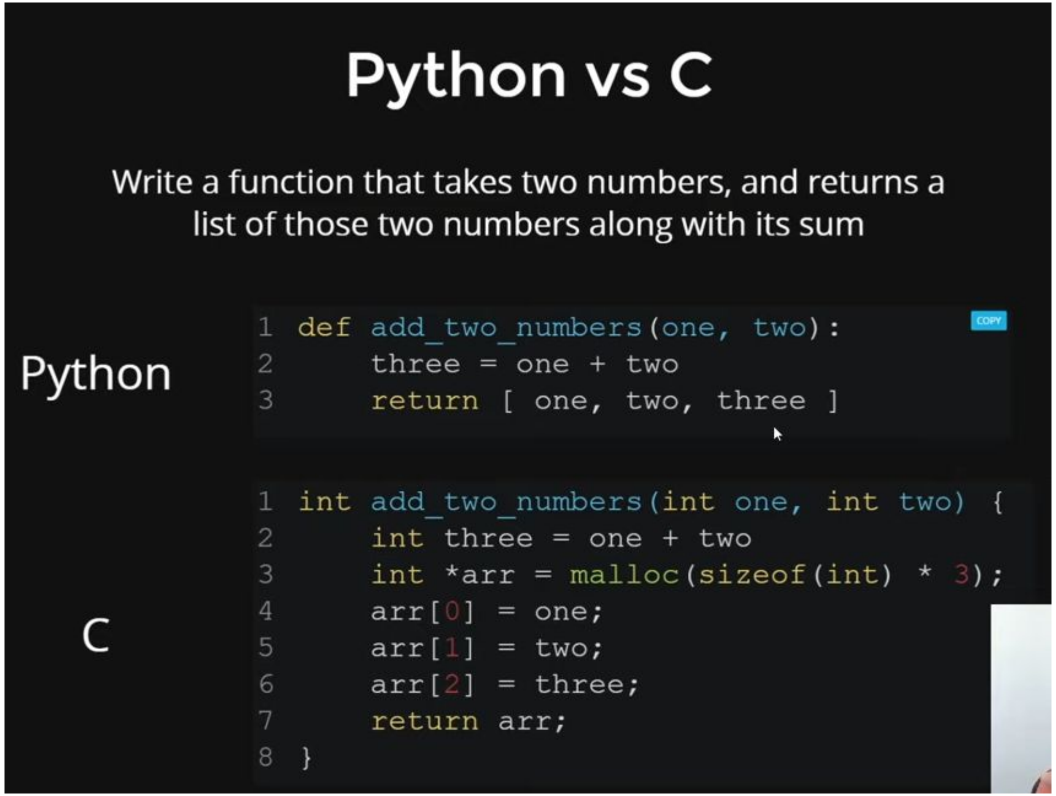
```
sandeep@NZXT-H440:~/UNSW/tutoring/cs1531/21t3/week02$ git add ignore2.py
sandeep@NZXT-H440:~/UNSW/tutoring/cs1531/21t3/week02$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   ignore2.py
```

After commit


Creating a snapshot / commit

```
sandeep@NZXT-H440:~/UNSW/tutoring/cs1531/21t3/week02$ git commit -m "preparing tute2 git slides"
[main 7bda684] preparing tute2 git slides
 1 file changed, 1 insertion(+)
 create mode 100644 cs1531/21t3/week02/ignore2.py
sandeep@NZXT-H440:~/UNSW/tutoring/cs1531/21t3/week02$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Week 1 Overview - Python



Python vs C

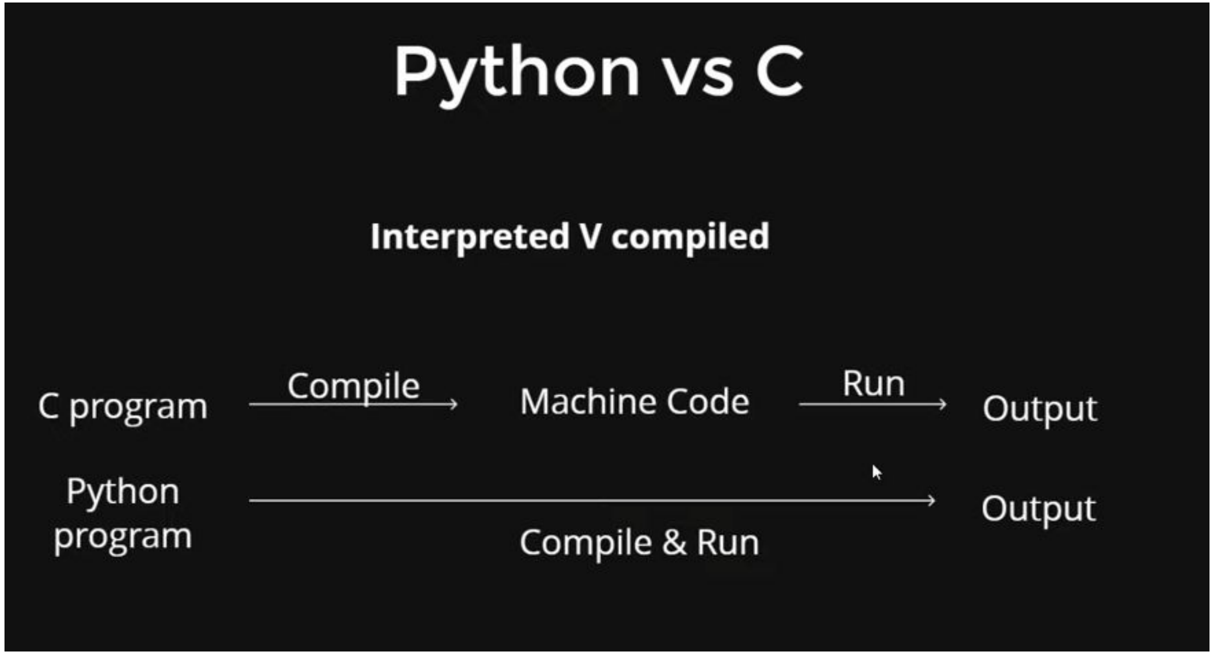
Write a function that takes two numbers, and returns a list of those two numbers along with its sum

Python

```python
def add_two_numbers(one, two):
    three = one + two
    return [ one, two, three ]
```

C

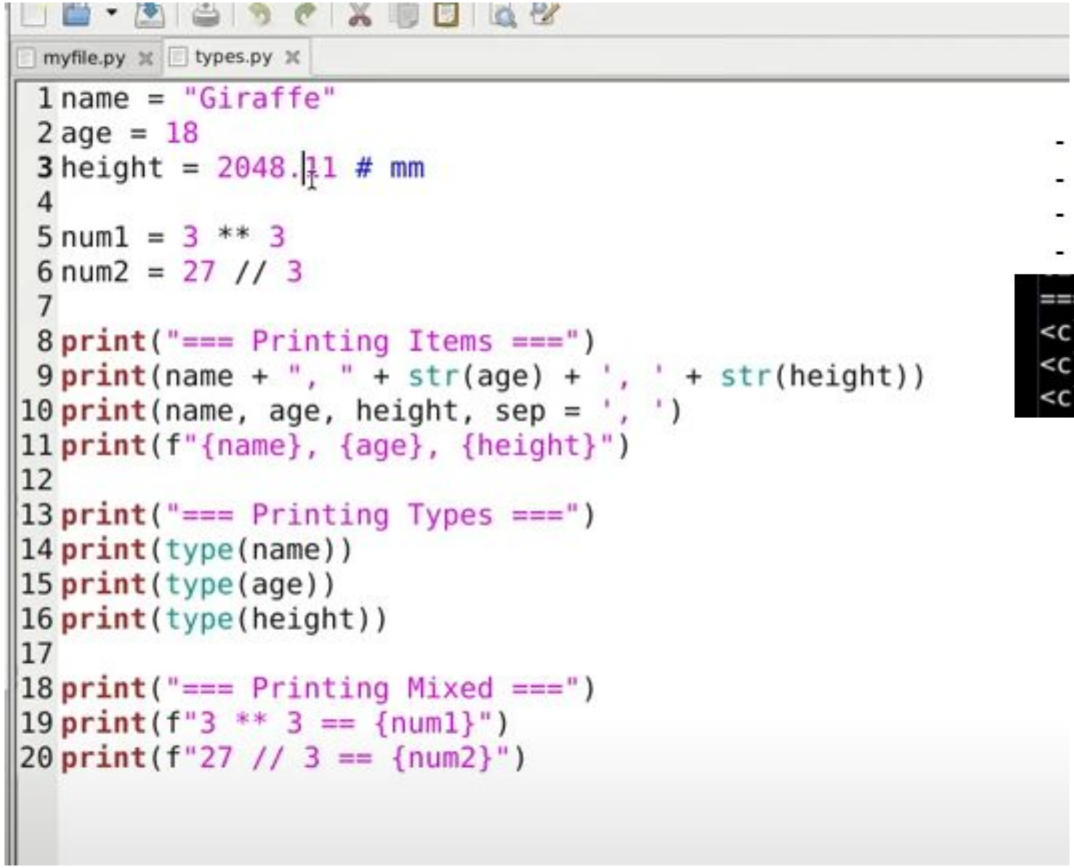
```c
int add_two_numbers(int one, int two) {
    int three = one + two
    int *arr = malloc(sizeof(int) * 3);
    arr[0] = one;
    arr[1] = two;
    arr[2] = three;
    return arr;
}
```



Python vs C

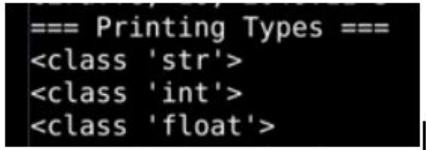
Interpreted V compiled

$$\begin{tikzcd}
\text{C program} \arrow[r, "\text{Compile}"] & \text{Machine Code} \arrow[r, "\text{Run}"] & \text{Output} \\
\text{Python program} \arrow[rr, "\text{Compile \& Run}"'] & & \text{Output}
\end{tikzcd}$$



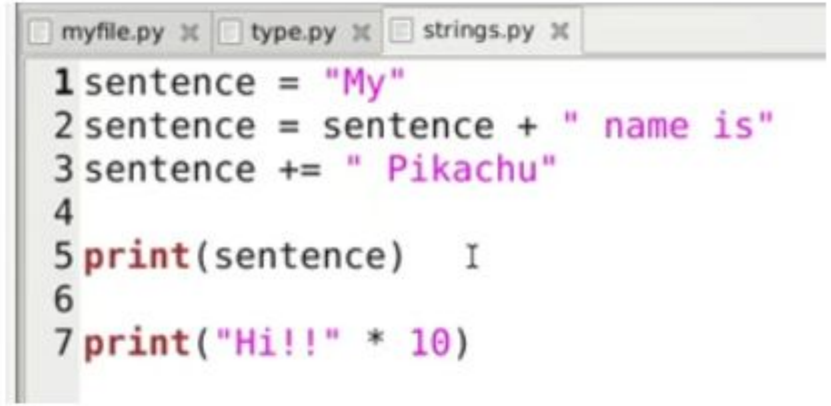
```
 1 name = "Giraffe"
 2 age = 18
 3 height = 2048.11 # mm
 4
 5 num1 = 3 ** 3
 6 num2 = 27 // 3
 7
 8 print("=== Printing Items ===")
 9 print(name + ", " + str(age) + ', ' + str(height))
10 print(name, age, height, sep = ', ')
11 print(f"{name}, {age}, {height}")
12
13 print("=== Printing Types ===")
14 print(type(name))
15 print(type(age))
16 print(type(height))
17
18 print("=== Printing Mixed ===")
19 print(f"3 ** 3 == {num1}")
20 print(f"27 // 3 == {num2}")
```

- Line 1) Python figures out types, don't have to specify
- Line 5) exponents, division
- Line 11) Use f strings
- Line 13)



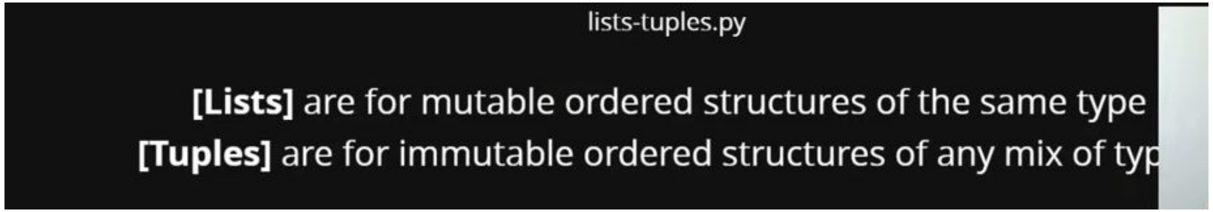
```
=== Printing Types ===
<class 'str'>
<class 'int'>
<class 'float'>
```

String manipulation



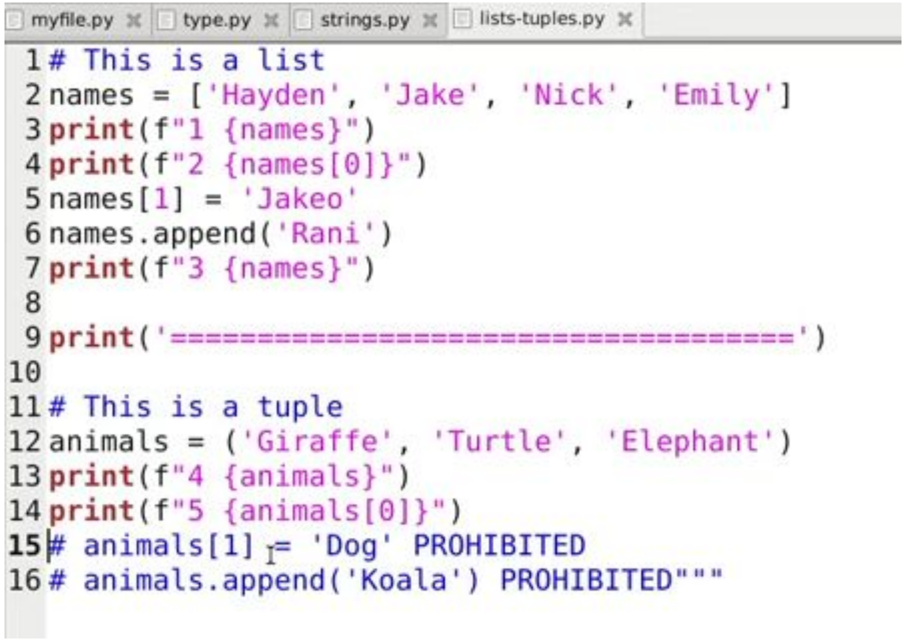
```python
sentence = "My"
sentence = sentence + " name is"
sentence += " Pikachu"

print(sentence)

print("Hi!!" * 10)
```

Python is really good for strings, if you have interview questions involving strings, you're going to die if you code it in C, hence why python is a popular language for interviews and leetcode

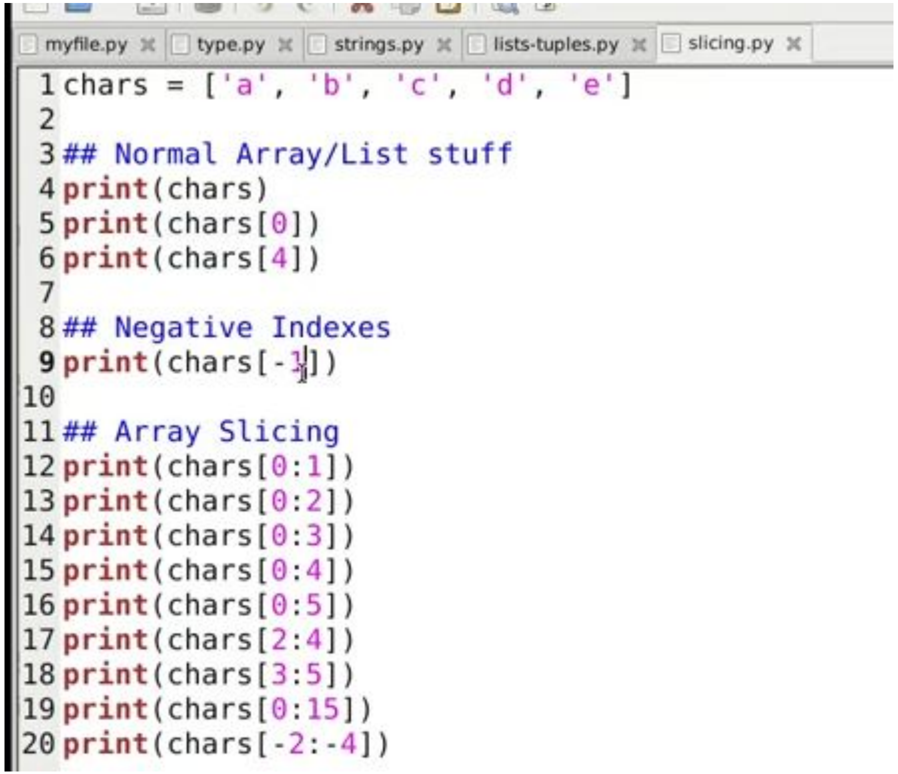


lists-tuples.py

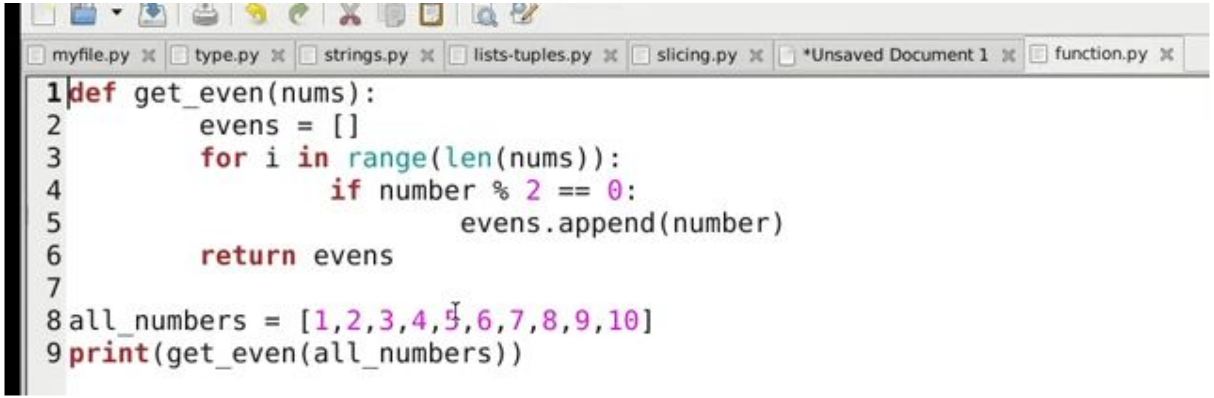
[Lists] are for mutable ordered structures of the same type
[Tuples] are for immutable ordered structures of any mix of typ



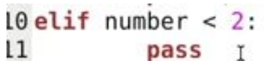
```python
# This is a list
names = ['Hayden', 'Jake', 'Nick', 'Emily']
print(f"1 {names}")
print(f"2 {names[0]}")
names[1] = 'Jakeo'
names.append('Rani')
print(f"3 {names}")

print('====================================')

# This is a tuple
animals = ('Giraffe', 'Turtle', 'Elephant')
print(f"4 {animals}")
print(f"5 {animals[0]}")
# animals[1] = 'Dog' PROHIBITED
# animals.append('Koala') PROHIBITED"""
```



```python
chars = ['a', 'b', 'c', 'd', 'e']

## Normal Array/List stuff
print(chars)
print(chars[0])
print(chars[4])

## Negative Indexes
print(chars[-1])

## Array Slicing
print(chars[0:1])
print(chars[0:2])
print(chars[0:3])
print(chars[0:4])
print(chars[0:5])
print(chars[2:4])
print(chars[3:5])
print(chars[0:15])
print(chars[-2:-4])
```

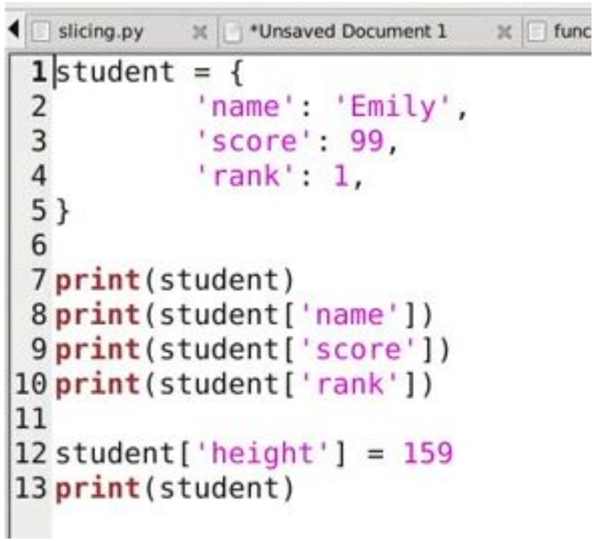


```python
def get_even(nums):
        evens = []
        for i in range(len(nums)):
                if number % 2 == 0:
                        evens.append(number)
        return evens

all_numbers = [1,2,3,4,5,6,7,8,9,10]
print(get_even(all_numbers))
```



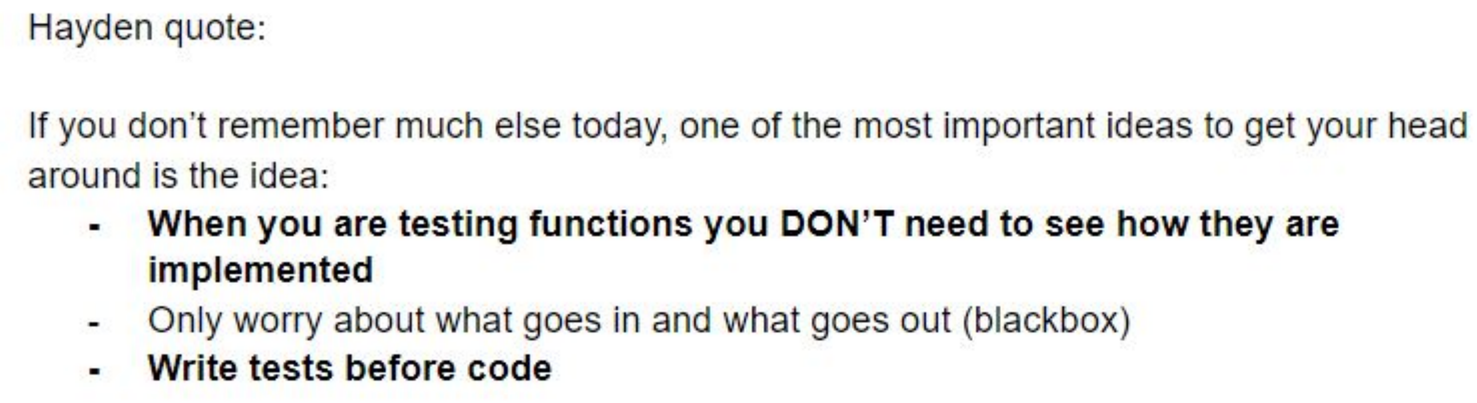
```python
elif number < 2:
        pass
```

Think of dictionaries like structs. You use them when you need a "collection" of items that are identified by a string description, rather than a numerical index (lists)



```python
student = {
        'name': 'Emily',
        'score': 99,
        'rank': 1,
}

print(student)
print(student['name'])
print(student['score'])
print(student['rank'])

student['height'] = 159
print(student)
```

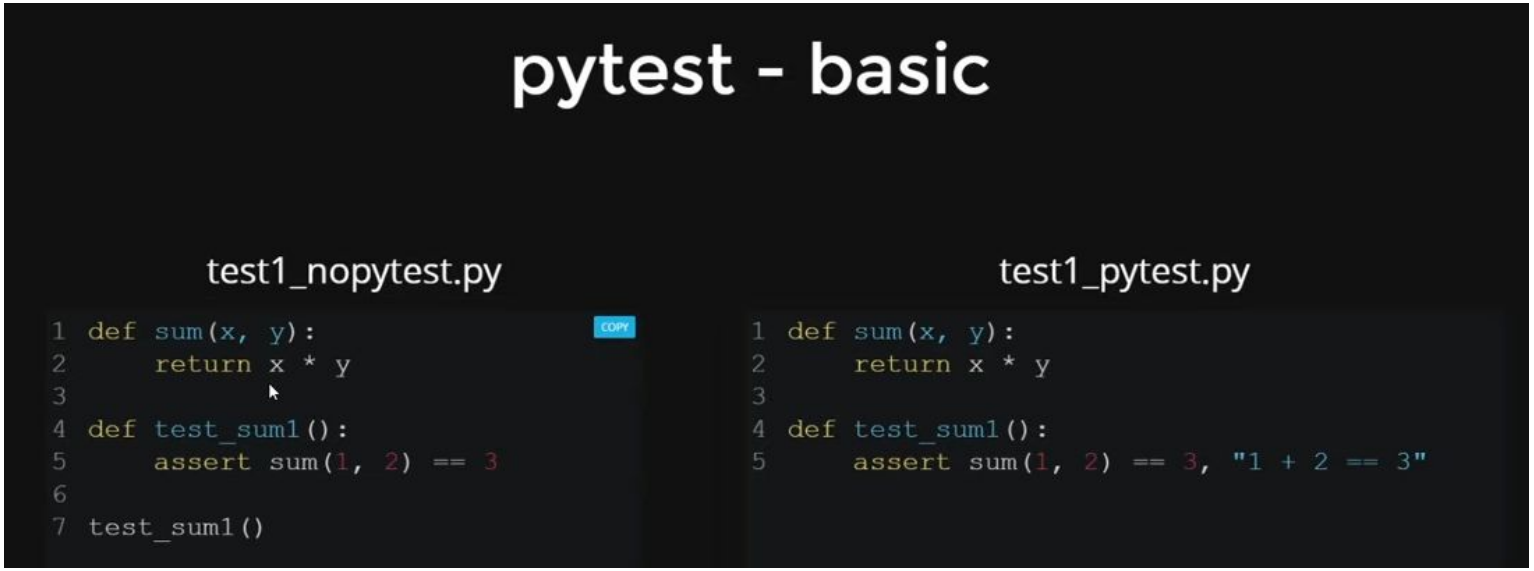
Week 1 Overview - Testing


Hayden quote:

If you don't remember much else today, one of the most important ideas to get your head around is the idea:

- **When you are testing functions you DON'T need to see how they are implemented**
- Only worry about what goes in and what goes out (blackbox)
- **Write tests before code**

This can be easily scaled



pytest - basic

test1_nopytest.py

```python
def sum(x, y):
    return x * y

def test_sum1():
    assert sum(1, 2) == 3

test_sum1()
```

test1_pytest.py

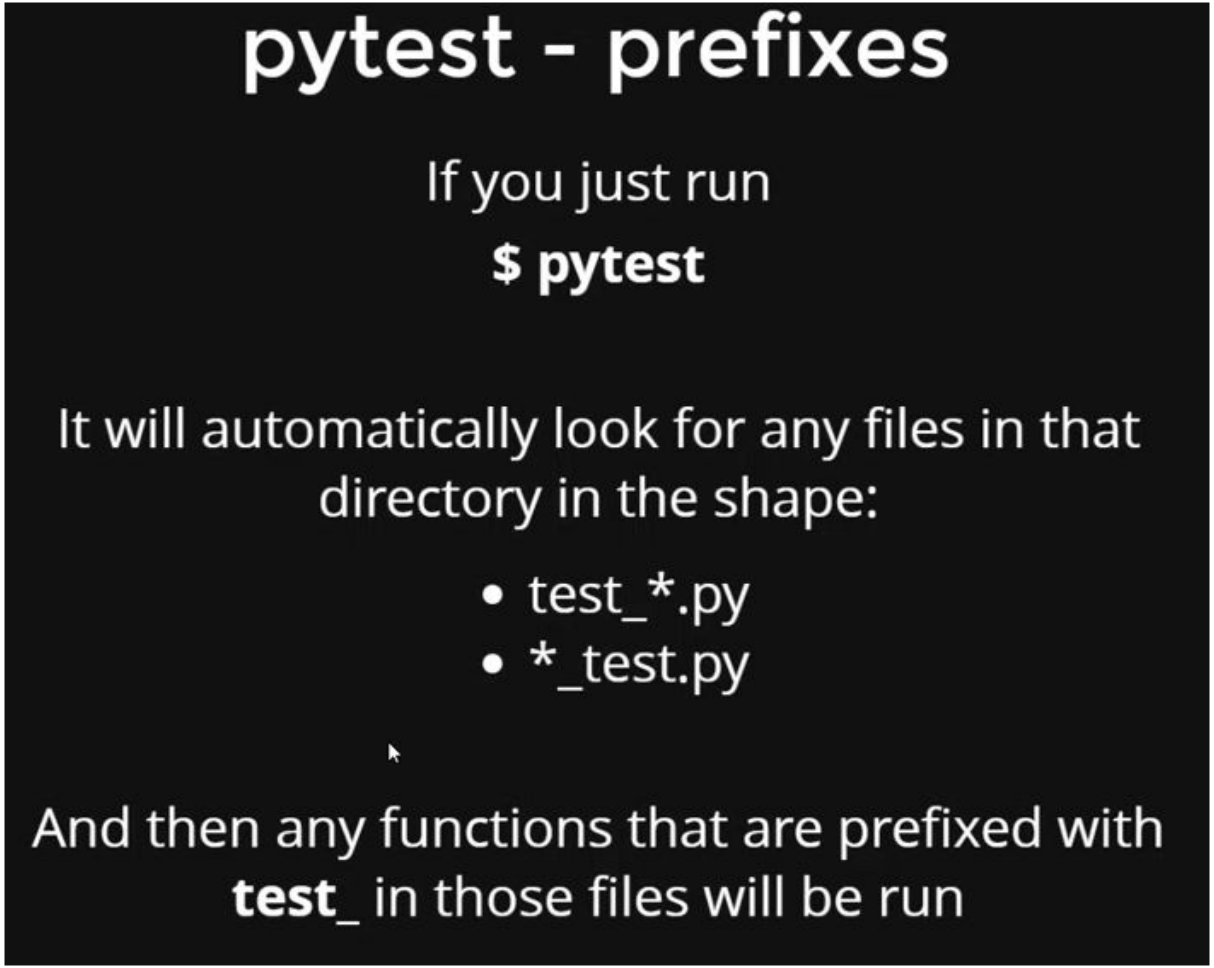
```python
def sum(x, y):
    return x * y

def test_sum1():
    assert sum(1, 2) == 3, "1 + 2 == 3"
```

pytest - more complicated

A more complicated test

test_multiple.py

```python
import pytest

def sum(x, y):
    return x + y

def test_small():
    assert sum(1, 2) == 3, "1, 2 == "
    assert sum(3, 5) == 8, "3, 5 == "
    assert sum(4, 9) == 13, "4, 9 == "

def test_small_negative():
    assert sum(-1, -2) == -3, "-1, -2 == "
    assert sum(-3, -5) == -8, "-3, -5 == "
    assert sum(-4, -9) == -13, "-4, -9 == "

def test_large():
    assert sum(84*52, 99*76) == 84*52 + 99*76, "84*52, 99*76 == "
    assert sum(23*98, 68*63) == 23*98 + 68*63, "23*98, 68*63 == "
```



pytest - prefixes

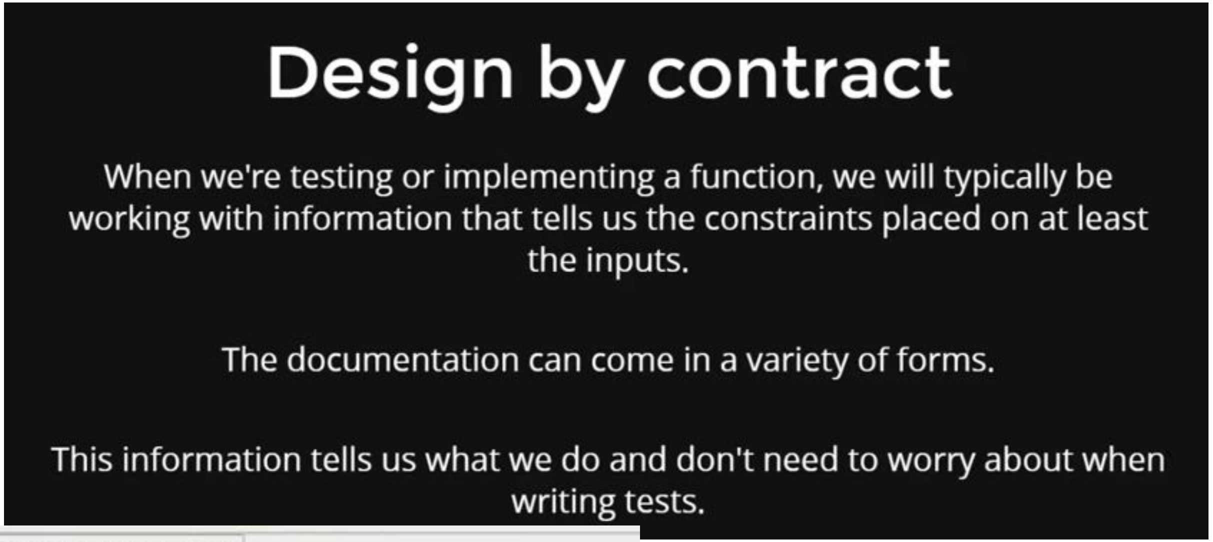
If you just run

$ pytest

It will automatically look for any files in that directory in the shape:

- test_*.py
- *_test.py

And then any functions that are prefixed with **test_** in those files will be run

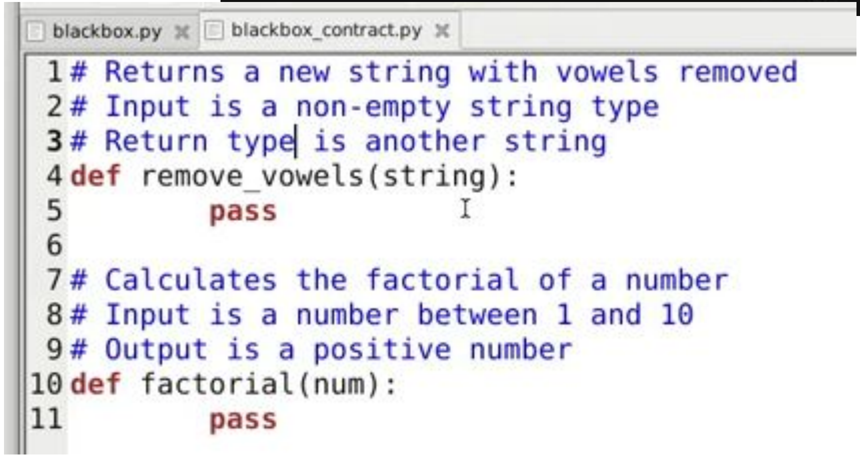


Design by contract

When we're testing or implementing a function, we will typically be working with information that tells us the constraints placed on at least the inputs.

The documentation can come in a variety of forms.

This information tells us what we do and don't need to worry about when writing tests.



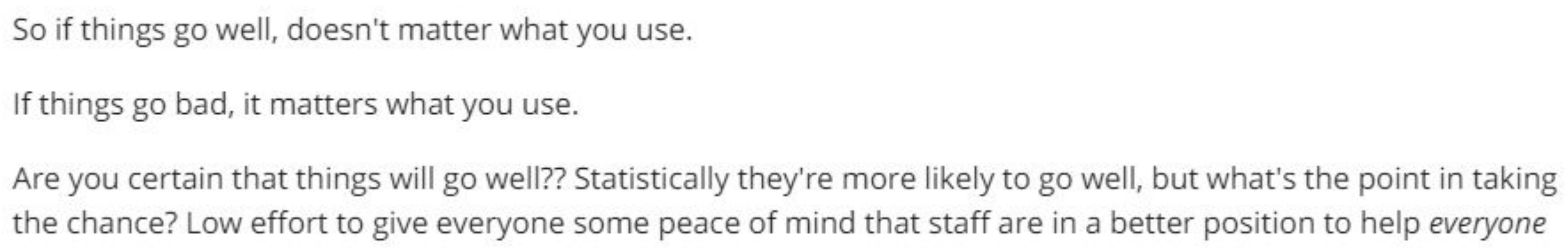
```python
# Returns a new string with vowels removed
# Input is a non-empty string type
# Return type is another string
def remove_vowels(string):
        pass

# Calculates the factorial of a number
# Input is a number between 1 and 10
# Output is a positive number
def factorial(num):
        pass
```

You don't need to test whether or not string input, or integer input is handled, it's assumed only string might be input for example

MS Teams

Do we need to use Microsoft Teams?#145

https://edstem.org/au/courses/7025/discussion/601012


So if things go well, doesn't matter what you use.

If things go bad, it matters what you use.

Are you certain that things will go well?? Statistically they're more likely to go well, but what's the point in taking the chance? Low effort to give everyone some peace of mind that staff are in a better position to help *everyone*

# At least keep meeting minutes! Discuss important things like allocation of work, etc.

# Message me if you don’t have access

Projects Out!

Tips:

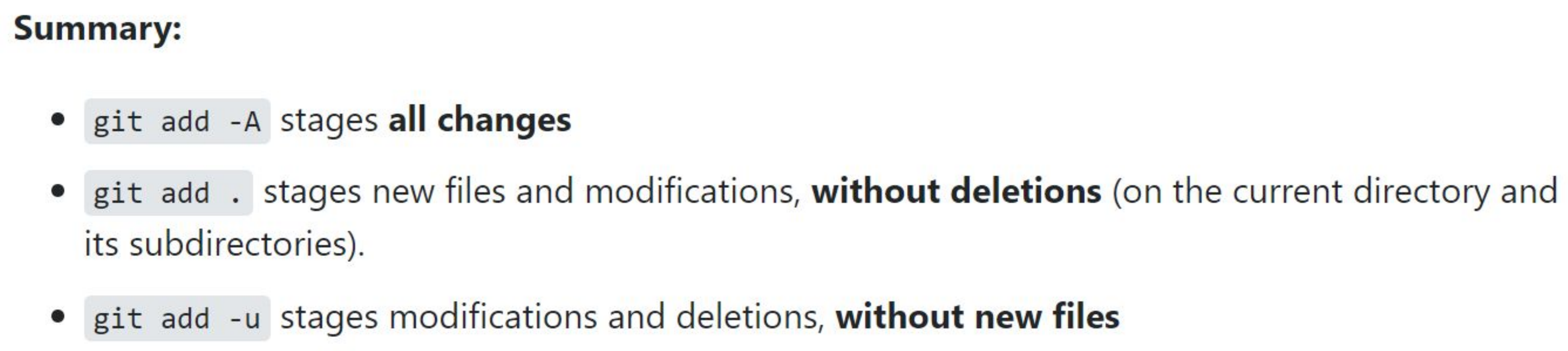
- Start early!
- Allocate workload today (I’ll come around and check)
- Have good git practices (avoids debugs from hell)
- Git pull often (syncs your local files, avoids conflicts)
- Have good branch names, like: sandeep-auth (makes it easier for me)
- Don’t delete branches (for marking purposes)
- Have good commit messages, like: “sandeep-auth: updated email function in auth_register”
- Merge into master only through gitlab’s web interface
- OVER COMMUNICATE

Git tips

AVOID: git *

Recommend: git adding each file individually if not completely comfortable

If you know what you're doing:

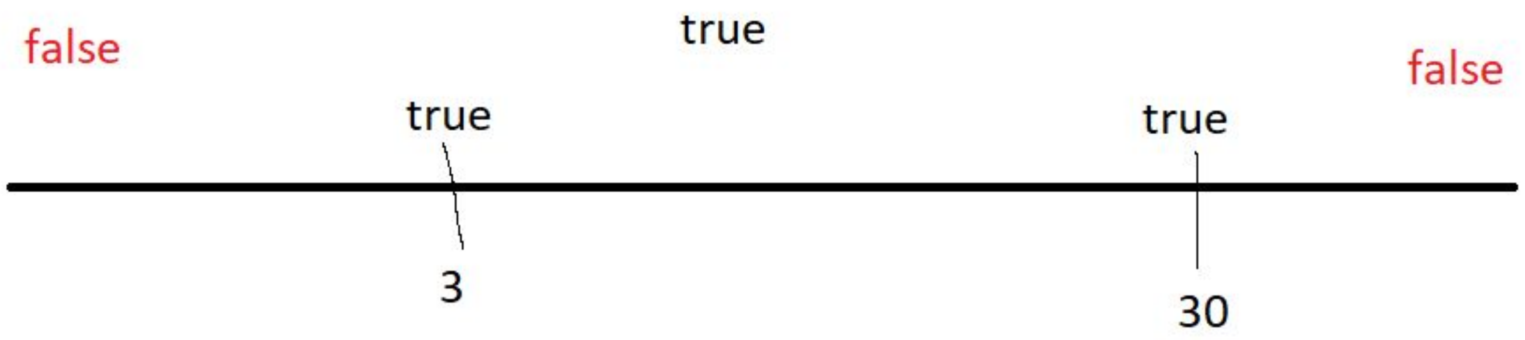


Summary:

- `git add -A` stages **all changes**
- `git add .` stages new files and modifications, **without deletions** (on the current directory and its subdirectories).
- `git add -u` stages modifications and deletions, **without new files**

Part B

FirstName


false
true
true
true
false
3
30