You can access these slides on the course Github:
https://github.com/natrask/ENM1050

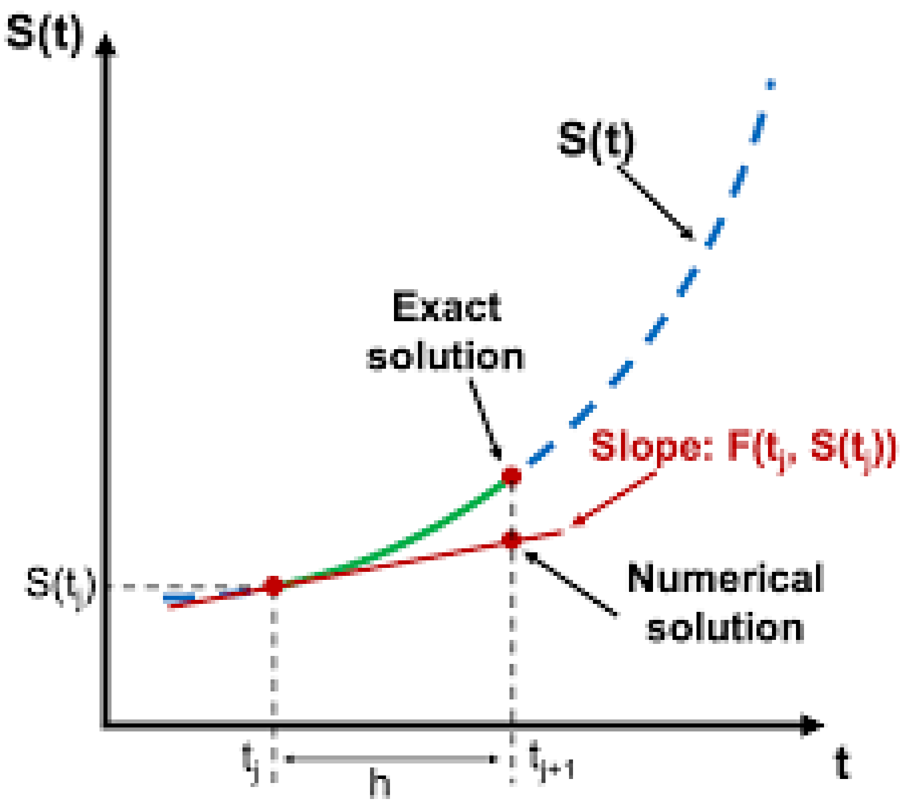
ENGR 1050
Intro to Scientific Computation

Lecture 06 – Classes and Solving Equations

Prof. Nat Trask
Mechanical Engineering & Applied Mechanics
University of Pennsylvania

Last time: `solving ODEs`

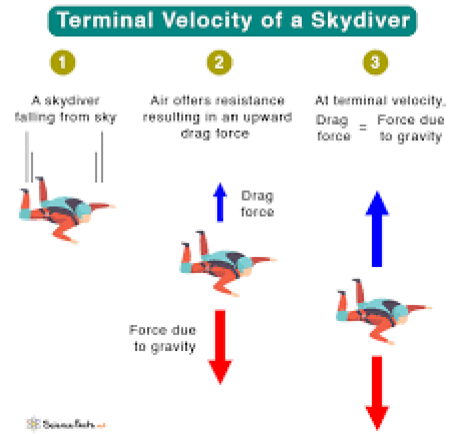
- We showed how we can use functions to encapsulate concepts, allowing us to write code at a higher level
- In last Wed class, we showed how to use a function to write a numerical ODE solver for an arbitrary problem by *encapsulating* the right hand side of the ODE
- In the HW due today, we needed to solve an optimization problem with an ODE solver embedded inside it.
- Encapsulating lets us split the optimization and ODE solve tasks

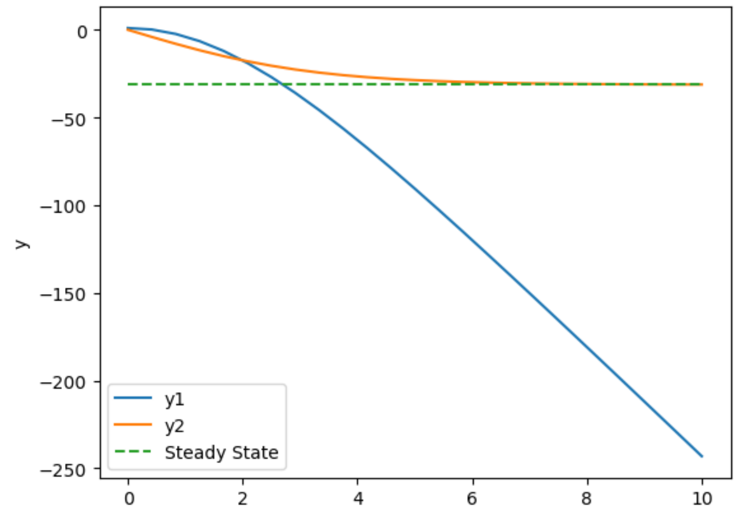


https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter22.03-The-Euler-Method.html

Last time: `libraries for ODEs`

- For smooth RHS problems we can use scipy
- <u>Always</u> develop a unit test to confirm the correctness of your implementation
- Take a look at the course github for some more information






```python
# Use SciPy to solve the skydiver problem

from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

alpha = -9.81  # acceleration due to gravity
beta = 0.01    # drag coefficient


def dydt(y, t):
    # Code to return RHS of ODE
    dy1dt = y[1]
    dy2dt = alpha + beta * y[1]**2
    return [dy1dt, dy2dt]

# Initial conditions
y0 = [1.0, 0.0]
t = np.linspace(0, 10, 25)
sol = odeint(dydt, y0, t)
```

✓ 1.6s

Python

Next couple weeks

- Last piece of python today!
- Building animations and interactive code
- Setting up your own environment – life beyond Jupyter notebooks!

- Next HW will be out Wed and due 10/9 (week from Wed, 2 more days)

Next couple weeks

- **Last building block of python today!**
- **To come: Building animations and interactive apps**
- **Setting up your own environment – life beyond Jupyter notebooks!**

- **Next HW will be out Wed and due 10/9 (week from Wed, 2 more days)**
- **We will be planning our quiz the following Monday (10/14)**
  - **Don’t stress!**
  - **Designed to make sure:**
    - **You’ve learned the basics of Python**
    - **You’re prepared to move forward with project-based work as the class progresses**
  - **There will be no computers used.**
  - **To prepare:**
    - **Make sure you have completed *all* in-class exercises**
    - **Make sure you are familiar with basic syntax (without autocomplete to help you)**
    - **Make sure you understand concepts (encapsulation, unit tests, style)**
    - **No need to memorize math – all formulas/definitions will be provided**
    - **Read python tutorial to get alternative explanations.**
    - **Office hours! Office hours! Office hours!**

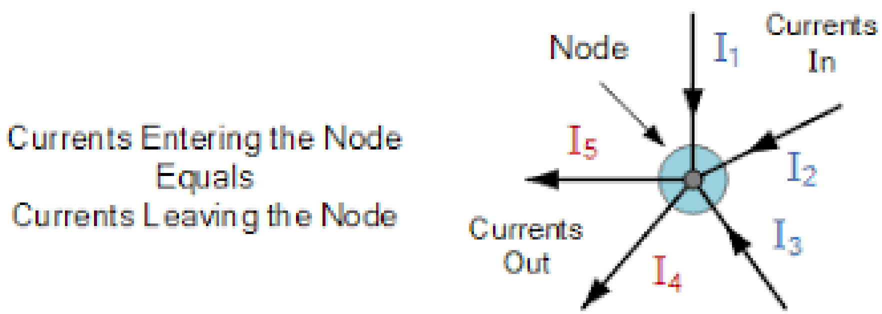
Today

Classes and Linear Systems of Equations

The governing equations of electrical circuits

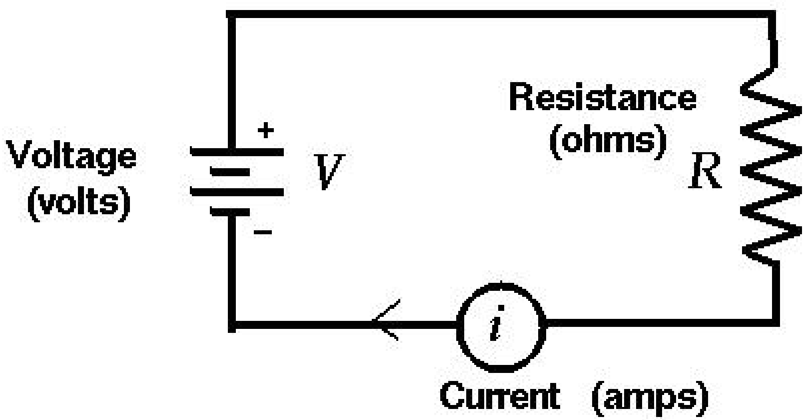
Don't worry! You don't need to have taken circuits, we'll just give you equations to solve

Kirchoffs current law



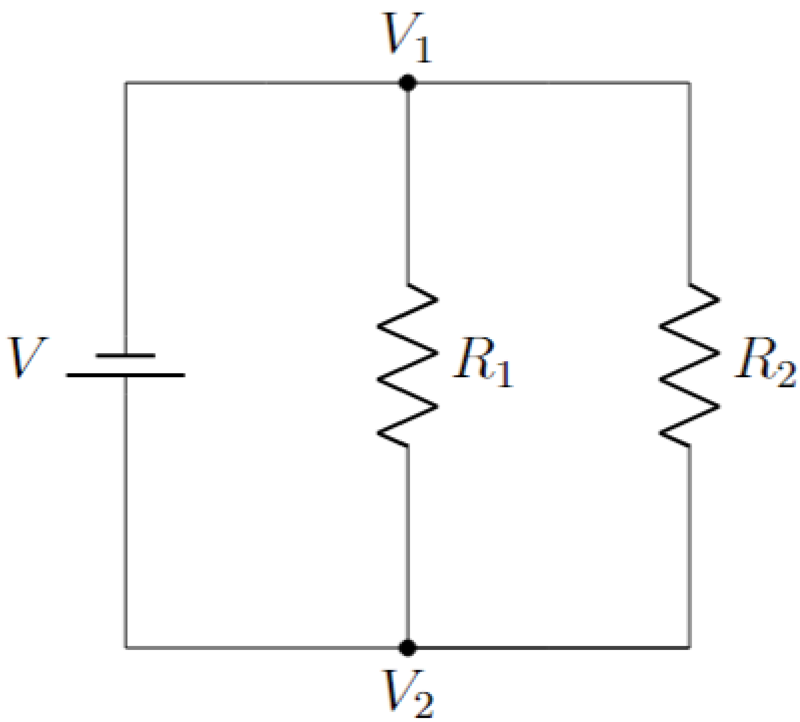
$$\sum_{\text{j touching i}} I_{ij} = 0$$

Ohms law



$$V = i\ R$$

Statics, circuits, and graphs



- Match voltage drop.

$$V_1 = V$$
$$V_2 = 0$$

- Ohm's law along each branch.

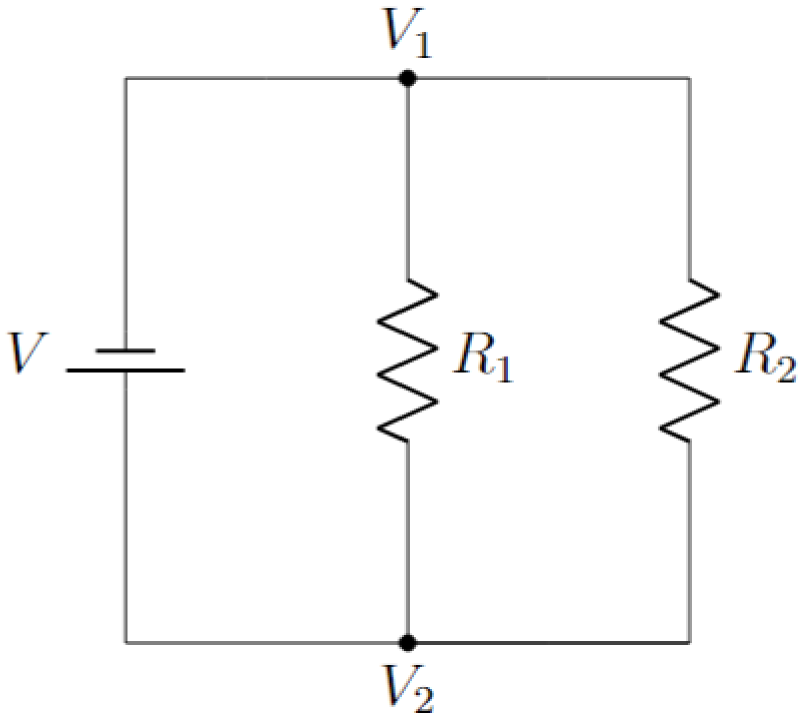
$$I_1 = \frac{1}{R_1}(V_2 - V_1)$$
$$I_2 = \frac{1}{R_2}(V_2 - V_1)$$

- Kirchoff's current law.

$$I_{total} = I_1 + I_2$$

We will take this and build it up to solver bigger resistor networks that can't be solved by hand

Electrical circuits



Kirchhoff's Current Law (KCL) Statement:

$$I_{total} = I_1 + I_2$$

Ohm's Law for each resistor:

$$I_1 = \frac{V}{R_1}$$

$$I_2 = \frac{V}{R_2}$$

Substitute Ohm's Law into KCL:

$$I_{total} = \frac{V}{R_1} + \frac{V}{R_2}$$

Factor out the voltage V :

$$I_{total} = V\left(\frac{1}{R_1} + \frac{1}{R_2}\right)$$

Alternatively, solve for the equivalent resistance R_{eq} :

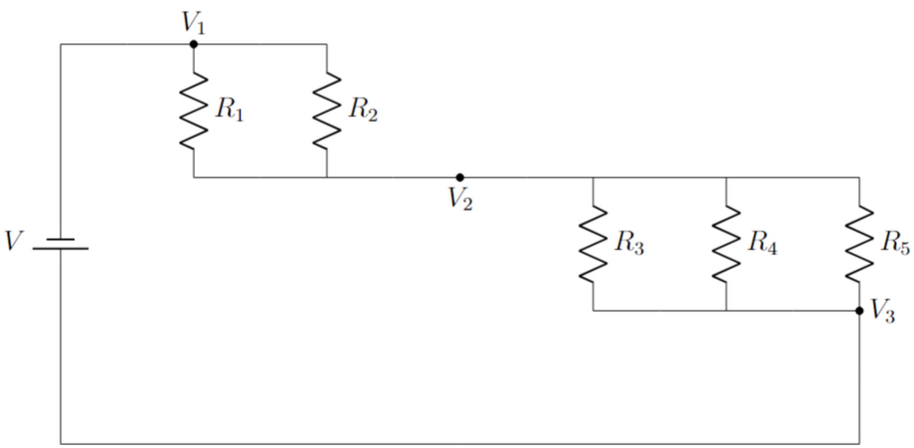
$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Express the total current in terms of the equivalent resistance:

$$I_{total} = \frac{V}{R_{eq}}$$

This is how you will learn to solve these kinds of equations in later classes. We will take this formula as a unit test to reproduce the total current.

A more complicated circuit (todays exercise)



- **Match voltage drop.**

$$V_1 = V$$

$$V_2 = 0$$

- **Kirchoff's current law.**

$$I_1 + I_2 = I_3 + I_4 + I_5$$

- **Ohm's law along each branch.**

$$I_1 = \frac{1}{R_1}(V_2 - V_1)$$

$$I_2 = \frac{1}{R_2}(V_2 - V_1)$$

$$I_3 = \frac{1}{R_3}(V_3 - V_2)$$

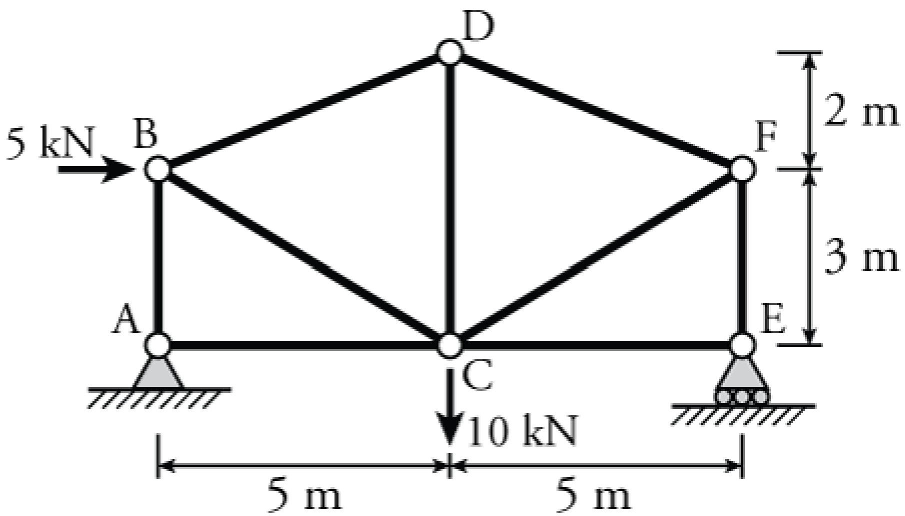
$$I_4 = \frac{1}{R_4}(V_3 - V_2)$$

$$I_5 = \frac{1}{R_5}(V_3 - V_2)$$

- **Total current.**

$$I_{total} = I_1 + I_2$$

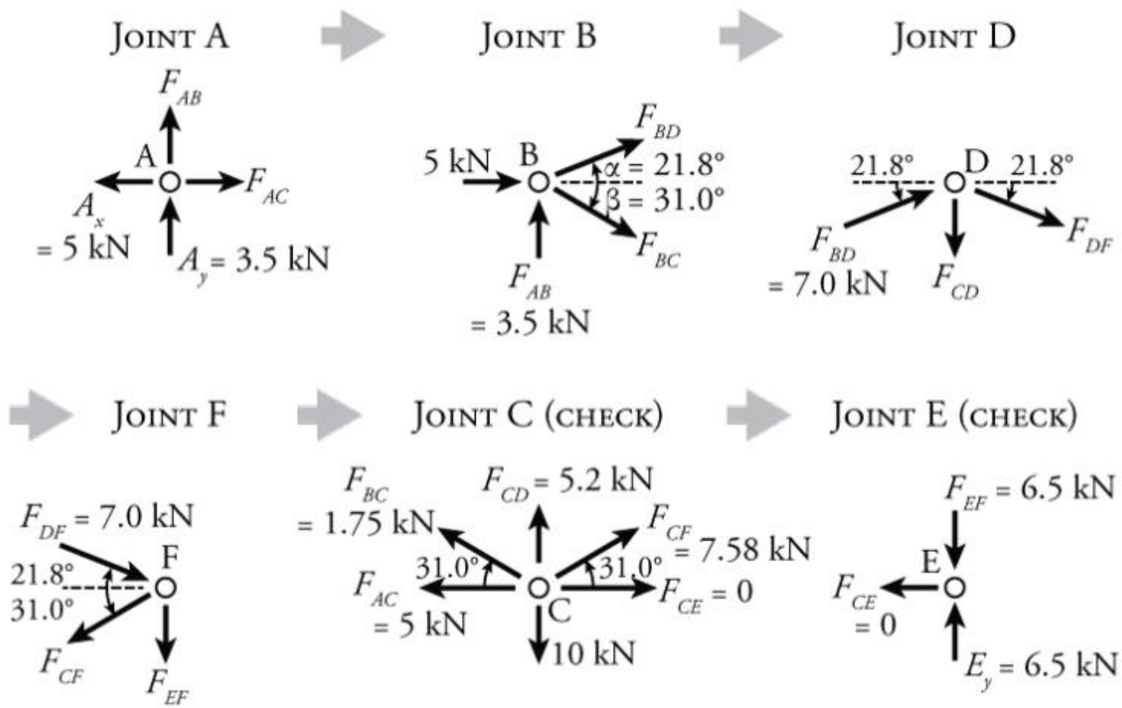
Not just electrical circuits! Structural mechanics



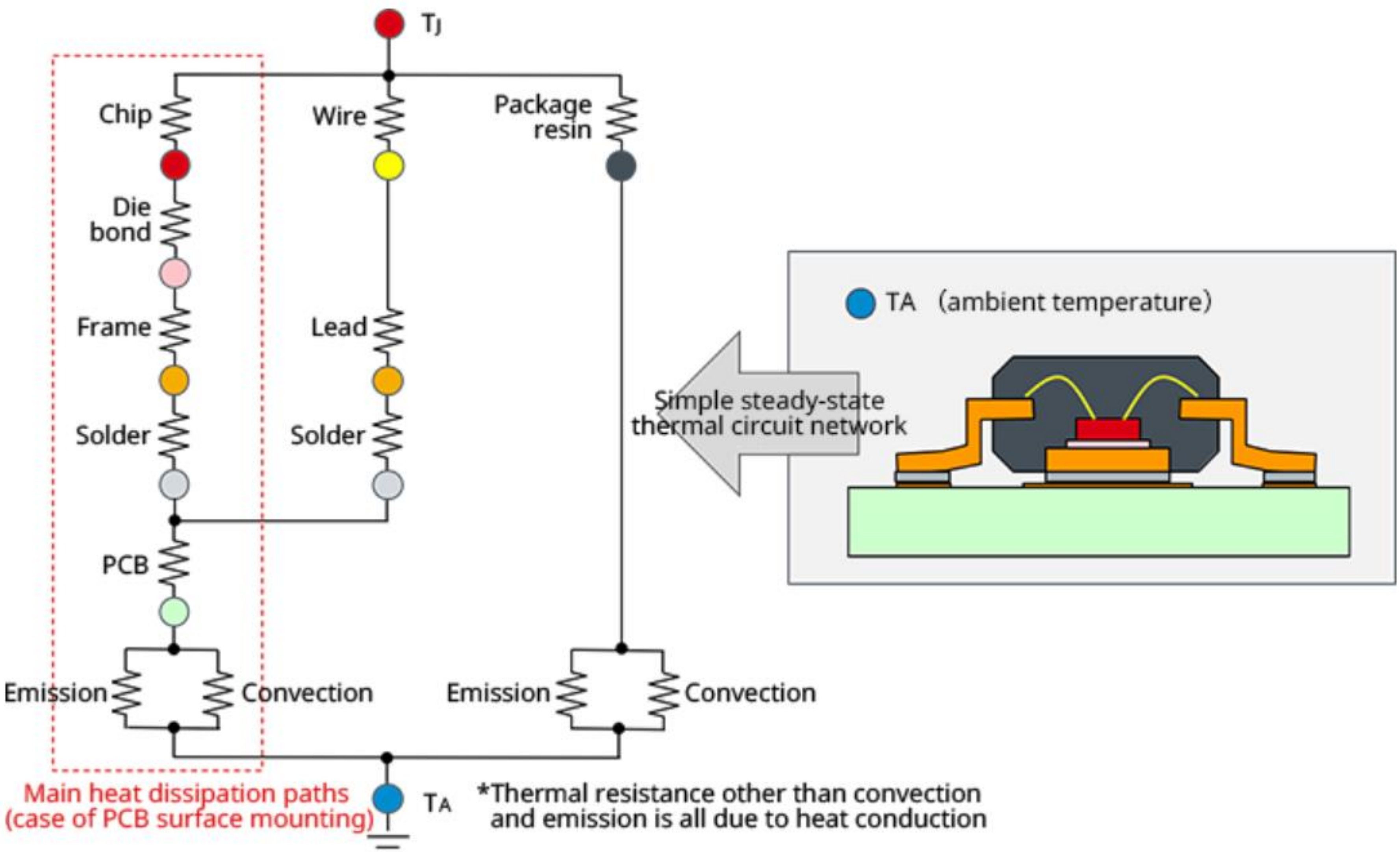
Compute axial forces in all truss members

$$\sum F_x = 0$$

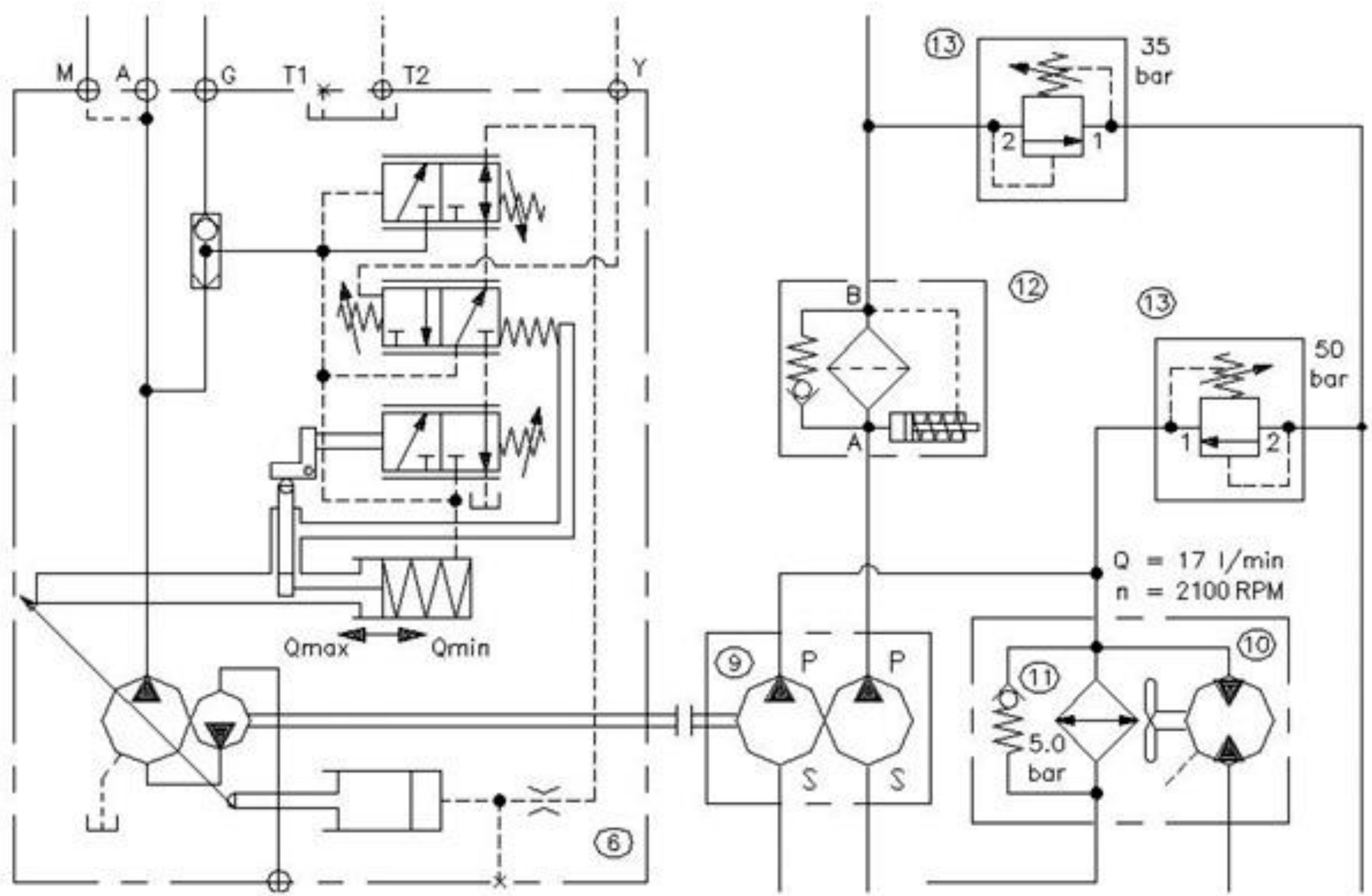
$$\sum F_y = 0$$



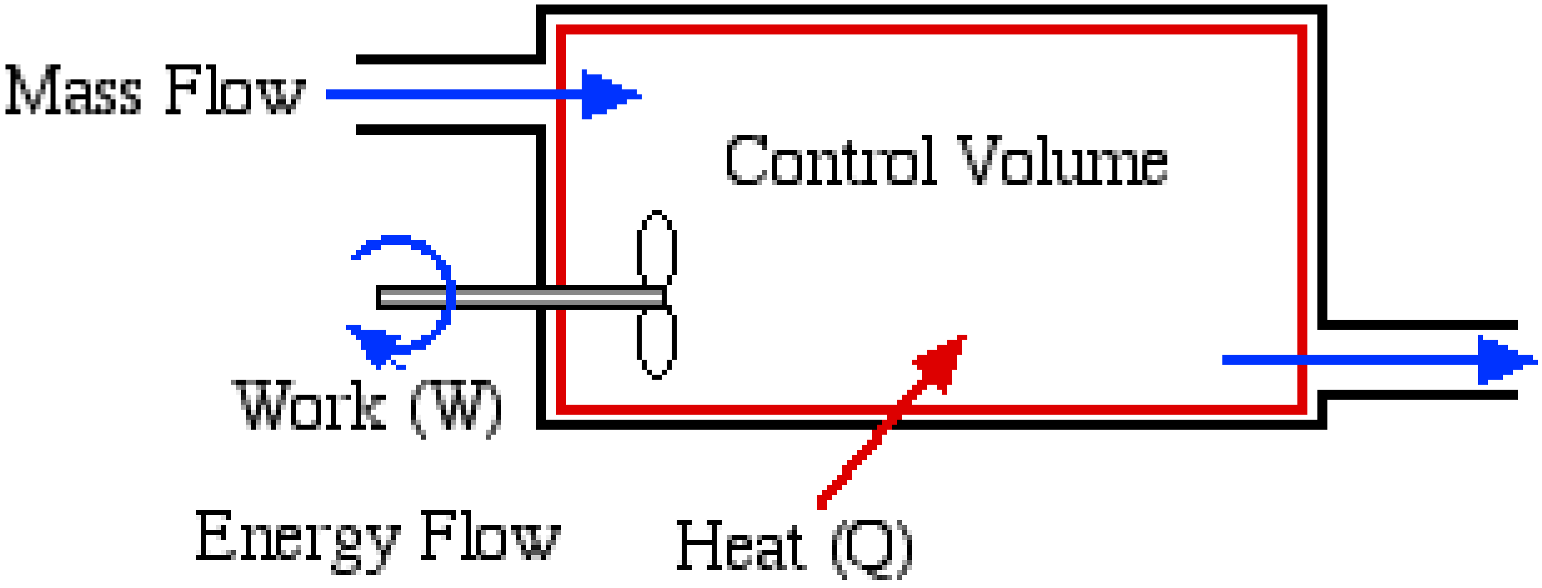
Not just electrical circuits! Heat transfer



Not just electrical circuits! Hydraulic circuits



Thermodynamics and control volumes



You'll learn how to derive all of these kinds of models if you take thermodynamics

This week

Solving these classes of problems on a computer

Two ingredients

Graphs

Linear Solvers

Graphs

An object consisting of:

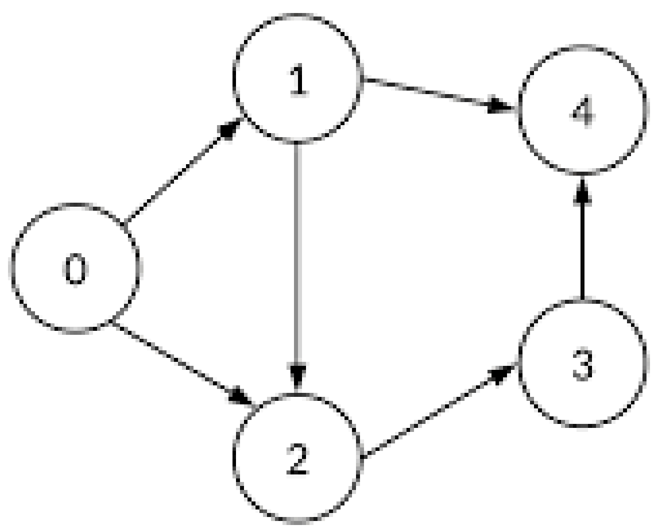
- **Nodes**/vertices
- **Edges** (+ direction)

The direction can be encoded in an **adjacency matrix**

For a given row:

If there's an edge pointing out

Add 1 to neighbors column



Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Solving systems of equations with linear algebra

$$
\begin{aligned}
3\,x_1 + 2\,x_2 + 7\,x_3 &= 1 \\
9\,x_1 + 2\,x_2 + 8\,x_3 &= 10 \\
2\,x_1 + 17\,x_2 + 9\,x_3 &= 6
\end{aligned}
\quad \Rightarrow \quad
\begin{bmatrix} 3 & 2 & 7 \\ 9 & 2 & 8 \\ 2 & 17 & 9 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
=
\begin{bmatrix} 1 \\ 10 \\ 6 \end{bmatrix}
$$

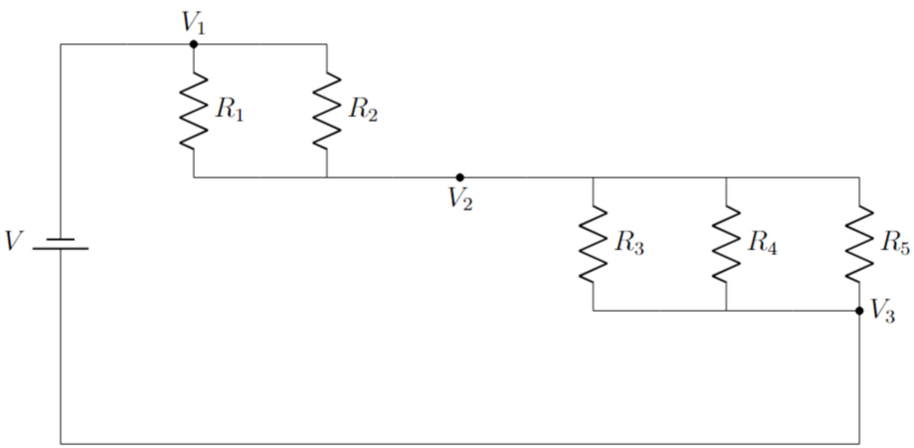
We'll learn more about linear algebra later.

For now, we can use the code to the right as a way to solve linear systems.

```python
import numpy as np

# Define the coefficient matrix
A = np.array([
    [3, 2, 7],
    [9, 2, 8],
    [2, 17, 9]
])

# Define the right-hand side vector
b = np.array([1, 10, 6])

# Solve the linear system
x = np.linalg.solve(A, b)

# Print the solution
print("Solution:", x)
```

Python

Ex1: Convert this into a matrix equation.



- **Match voltage drop.**

$$V_1 = V$$

$$V_2 = 0$$

- **Kirchoff's current law.**

$$I_1 + I_2 = I_3 + I_4 + I_5$$

- **Ohm's law along each branch.**

$$I_1 = \frac{1}{R_1}(V_2 - V_1)$$

$$I_2 = \frac{1}{R_2}(V_2 - V_1)$$

$$I_3 = \frac{1}{R_3}(V_3 - V_2)$$

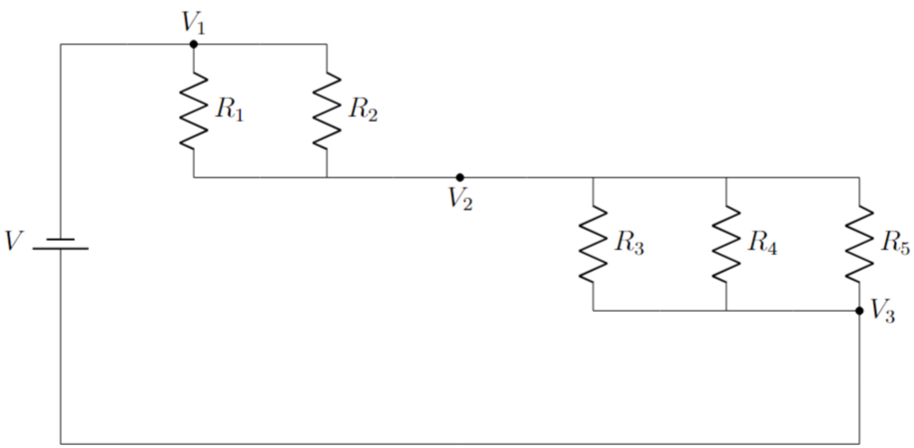
$$I_4 = \frac{1}{R_4}(V_3 - V_2)$$

$$I_5 = \frac{1}{R_5}(V_3 - V_2)$$

- **Total current.**

$$I_{total} = I_1 + I_2$$

Ex1: Convert this into a matrix equation.



- **Match voltage drop.**

$$V_1 = V$$
$$V_2 = 0$$

- **Kirchoff's current law.**

$$I_1 + I_2 = I_3 + I_4 + I_5$$

- **Ohm's law along each branch.**

$$I_1 = \frac{1}{R_1}(V_2 - V_1)$$
$$I_2 = \frac{1}{R_2}(V_2 - V_1)$$
$$I_3 = \frac{1}{R_3}(V_3 - V_2)$$
$$I_4 = \frac{1}{R_4}(V_3 - V_2)$$
$$I_5 = \frac{1}{R_5}(V_3 - V_2)$$

- **Total current.**

$$I_{total} = I_1 + I_2$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1} & 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{R_3} & \frac{1}{R_3} & 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{R_4} & \frac{1}{R_4} & 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{R_5} & \frac{1}{R_5} & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The Last Piece of Python…

Last time

Functions: Generalization, Maintenance and Encapsulation

Why use functions? Generalization

Same code can be used more than once with parameters to allow for differences

BEFORE

```
diameter_large = 2.54 * 1.65
print('Large ball: ', diameter_large, 'cm')

diameter_med = 2.54 * 1.01
print('Medium ball: ', diameter_med, 'cm')

diameter_small = 2.54 + 0.46
print('Small ball: ', diameter_small, 'cm')
```

would not have made this error

AFTER

```
def print_as_cm(inches, name):
    cm = 2.54 * inches
    print(name, ':', cm, 'cm')

print_as_cm(1.65, 'Large ball')
print_as_cm(1.01, 'Medium ball')
print_as_cm(0.46, 'Small ball')
```

only type these lines once

Why use functions? Maintenance

Much easier to make changes

BEFORE

```
diameter_large = 2.54 * 1.65
print('Large ball: ', diameter_large, 'cm')

diameter_med = 2.54 * 1.01
print('Medium ball: ', diameter_med, 'cm')

diameter_small = 2.54 + 0.46
print('Small ball: ', diameter_small, 'cm')
```

AFTER

```
def print_as_cm(inches, name):
    cm = 2.54 * inches
    print(name, ':', cm, 'cm')

print_as_cm(1.65, 'Large ball')
print_as_cm(1.01, 'Medium ball')
print_as_cm(0.46, 'Small ball')
```

Can change to 'centimeter' with only one change

Why use functions? Encapsulation

Much easier to debug!

BEFORE

```
diameter_large = 2.54 * 1.65
print('Large ball: ', diameter_large, 'cm')

diameter_med = 2.54 * 1.01
print('Medium ball: ', diameter_med, 'cm')

diameter_small = 2.54 + 0.46
print('Small ball: ', diameter_small, 'cm')
```

What are we doing here?

AFTER

```
def print_as_cm(inches, name):
    cm = 2.54 * inches
    print(name, ':', cm, 'cm')

print_as_cm(1.65, 'Large ball')
print_as_cm(1.01, 'Medium ball')
print_as_cm(0.46, 'Small ball')
```

Oh, printing as centimeters!

Introducing classes

A **class** is a Python object which encapsulates a collection of functions and variables to help structure and organize your code.

```
class className:
    <statement_1>
    …
    <statement_N>
```

Ex:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f():
        return 'hello world’
```

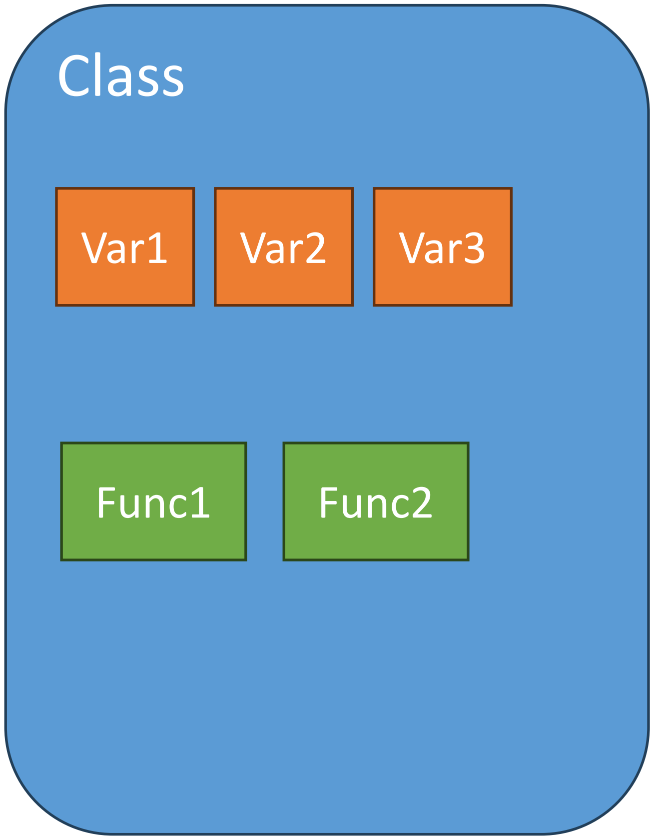
```
print(MyClass.i)
MyClass.f()
```

[11] ✓ 0.0s Python

```
12345

'hello world'
```

https://docs.python.org/3/tutorial/classes.html

Diagram of a class – simplest setting




```python
class MyClass:
    """A simple example class"""
    i = 12345

    def f():
        return "hello world"
```

Python

```python
print(MyClass.i)
MyClass.f()
```

[11] 0.0s Python

```
12345

'hello world'
```

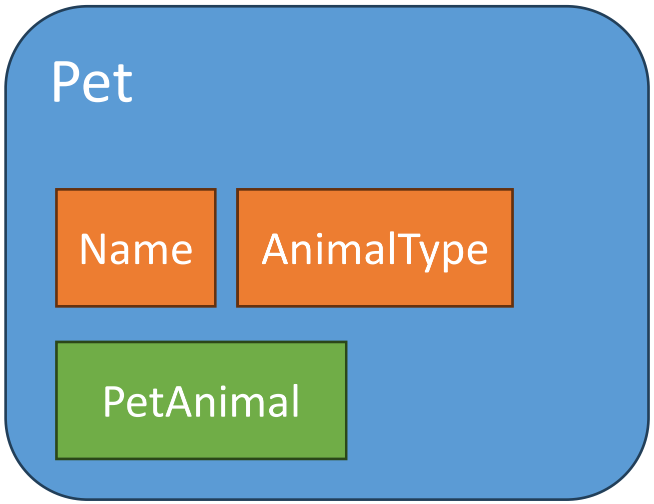
Just a container for holding variables and functions

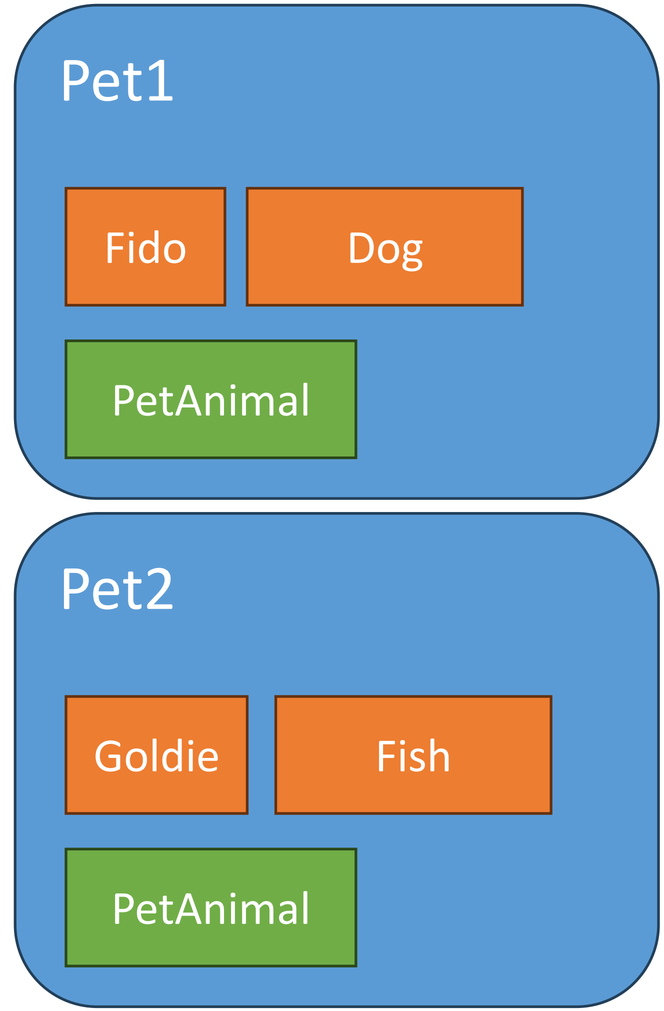
Polymorphism: Instantiation and *self*

```
class className:
    classvar1 = value # all members of class share this
    …
    classvarN = value # all members of class share this

    def __init__(self, input1, …, inputN):
      self.var1 = input1 # specialized var
      …
      self.varN = inputN # specialized var

    def myfunc(self, func_input):
      <statement>
```

That seems complicated, we'll go through with examples.
Actual syntax is in today's exercise.

Diagram of a class – inheritance





Pet1 = Pet('Fido','Dog')
Pet2 = Pet('Goldie','Fish')
Pet1.PetAnimal()
- Good boy Fido, have a treat!

Pet2.PetAnimal()
- My hand got wet trying to pet Goldie.

Allows specialization of behavior to different contexts

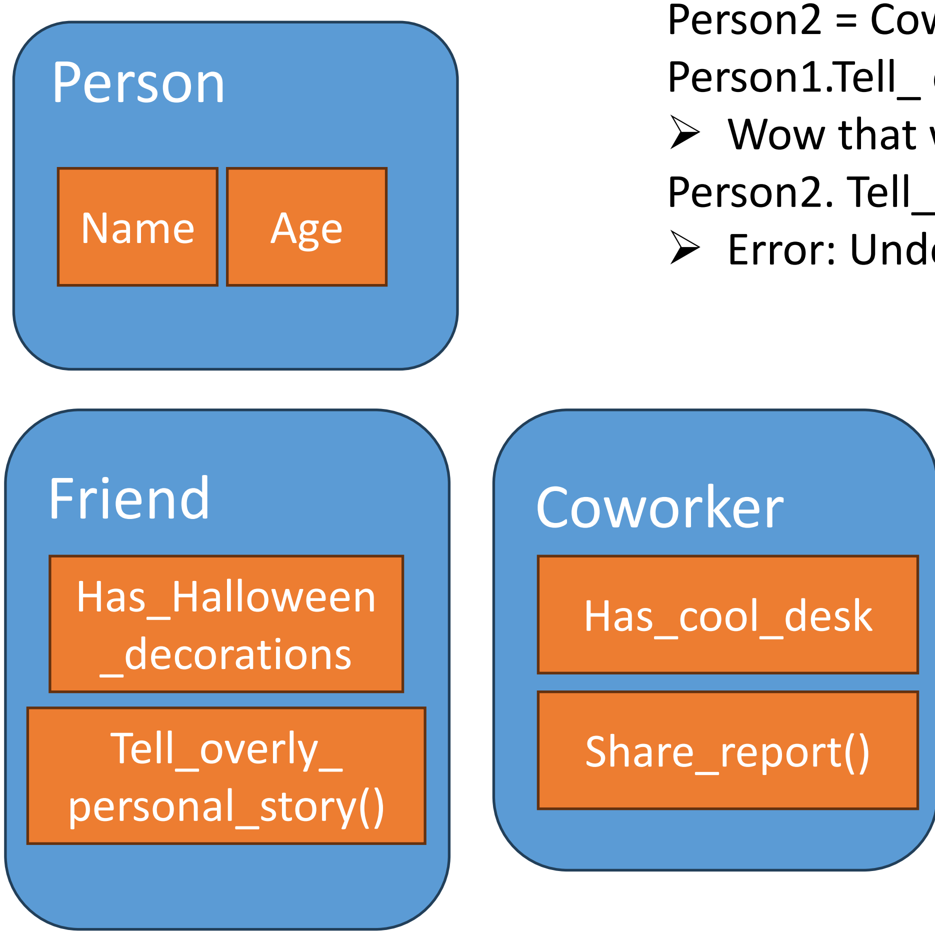
Inheritance

Person1 = Friend(‘Bob’,25,False)
Person2 = Coworker(‘Nancy’,26,True)
Person1.Tell_ overly_personal_story()

- ➢ Wow that was embarrassing, but funny!

Person2. Tell_overly_personal_story()

- ➢ Error: Undefined, why would you do that?



Allows specialization of generic behavior into contexts

Syntax

```python
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    def whatKindofAnimal():
        return f"I am a dog!"

    def bark(self):
        return f"{self.name} says woof!"

    def get_info(self):
        return f"Name: {self.name}, Age: {self.age}, Breed: {self.breed}"
```

Specify parent class

Specify how to initialize parent variables

```python
class WorkingDog(Dog):
    def __init__(self, name, age, breed, job):
        super().__init__(name, age, breed)
        self.job = job

    def do_job(self):
        return f"{self.name} is doing their job: {self.job}"

# Example usage:
working_dog = WorkingDog("Max", 4, "German Shepherd", "Police Dog")
print(working_dog.bark())
print(working_dog.get_info())
print(working_dog.do_job())
```

[ ] Python

In-Class 09: Classes and linear solvers

Do this with a partner.

Turn in as a pair on Canvas.

Tips for pair programming:

- Switch off who is typing.
- The person who is not typing should:
  - Make comments or suggest potential solutions
  - Be “devil’s advocate”: what are potential issues with what is being typed
  - Suggest other things to explore

At-Home: Next assignment will be released Wed.