Modules & Objects

ÉCOLE NORMALE SUPÉRIEURE DE LYON

Ali Farnudi, Fall 2021

Programming paradigms

In Python

How you write/organise your code

Modular design

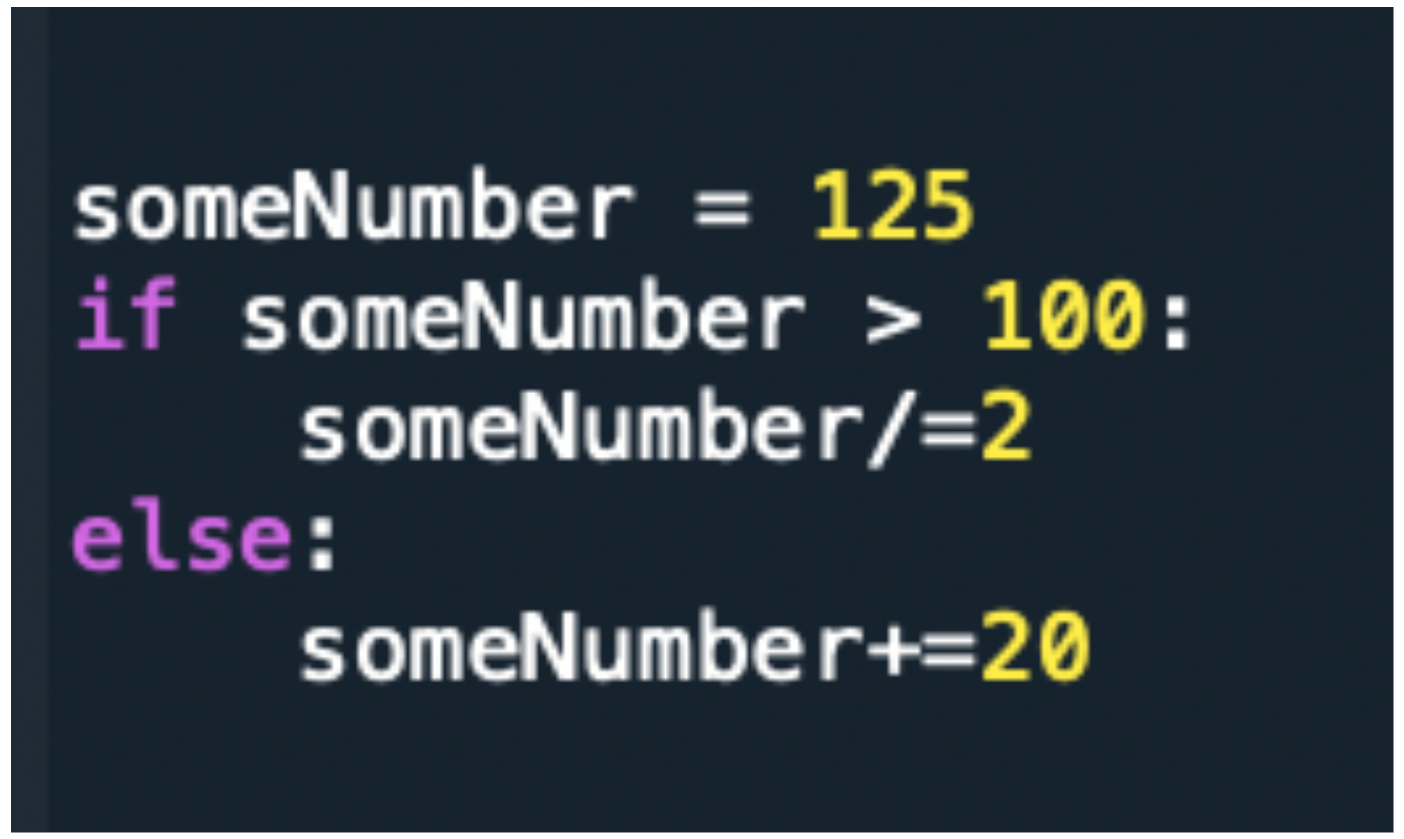
- Structured/Procedural
- Object Oriented Programming
- Functional Programming

Best for projects that scale out.

Programming paradigms

Structured/Procedural

- Hard to read and lengthy
- Hard to reuse (copy/pate)



```
someNumber = 125
if someNumber > 100:
    someNumber/=2
else:
    someNumber+=20
```

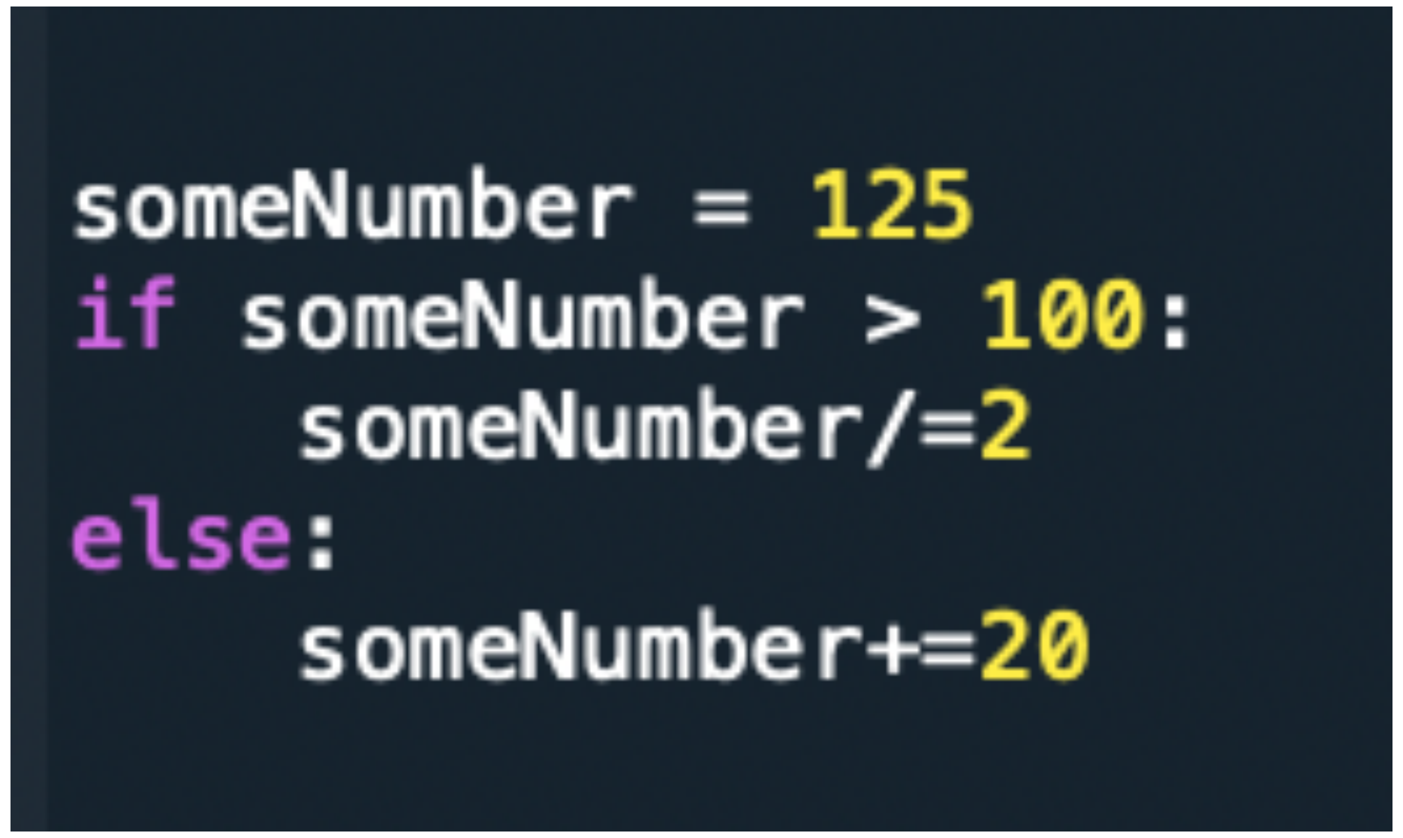
OO

Functional

Programming paradigms

Structured/Procedural

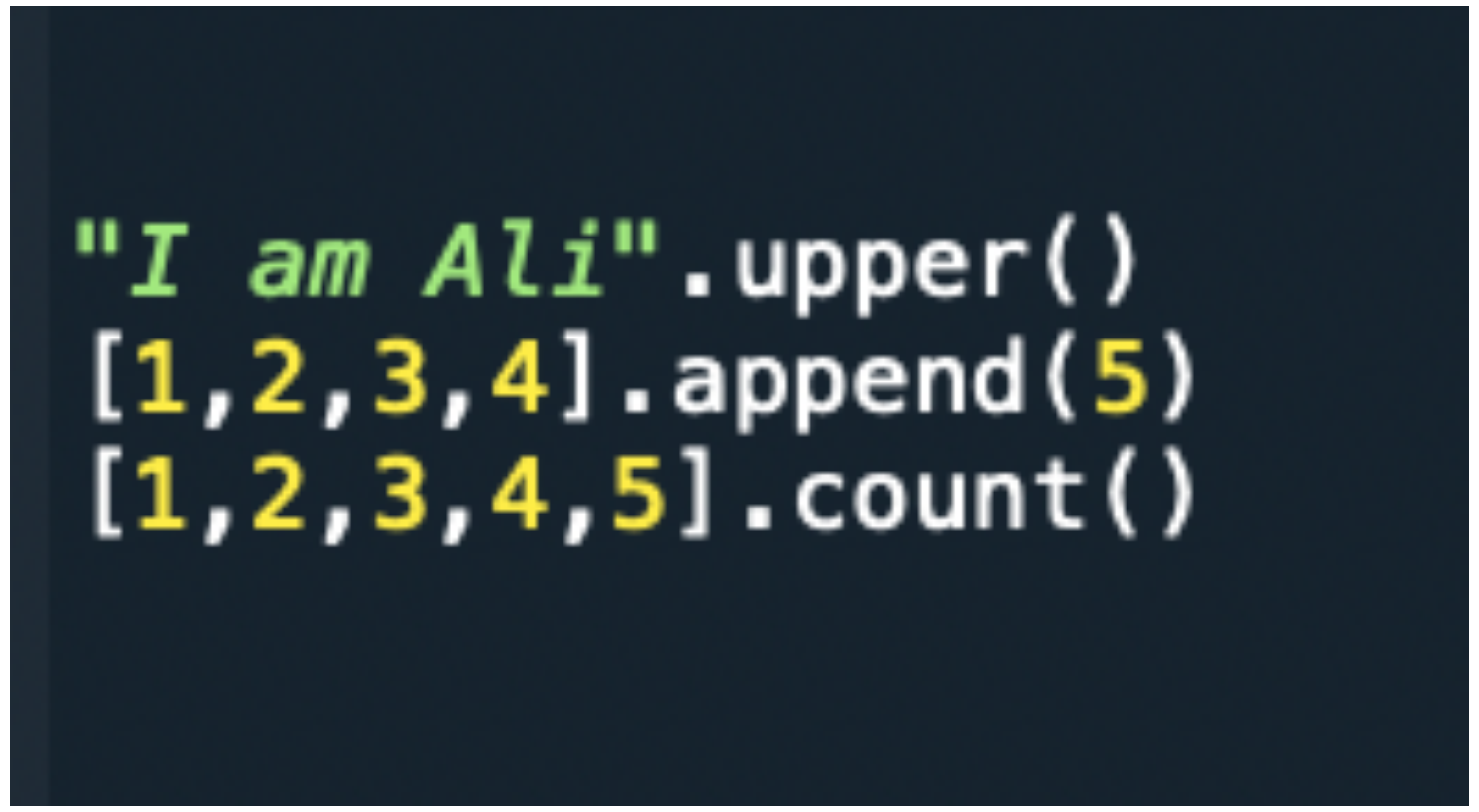
- Hard to read and lengthy
- Hard to reuse (copy/ pate)



```
someNumber = 125
if someNumber > 100:
    someNumber/=2
else:
    someNumber+=20
```

OO

- Smaller chunks (objects)
- If something breaks we go to the object
- Just worry about how objects interact



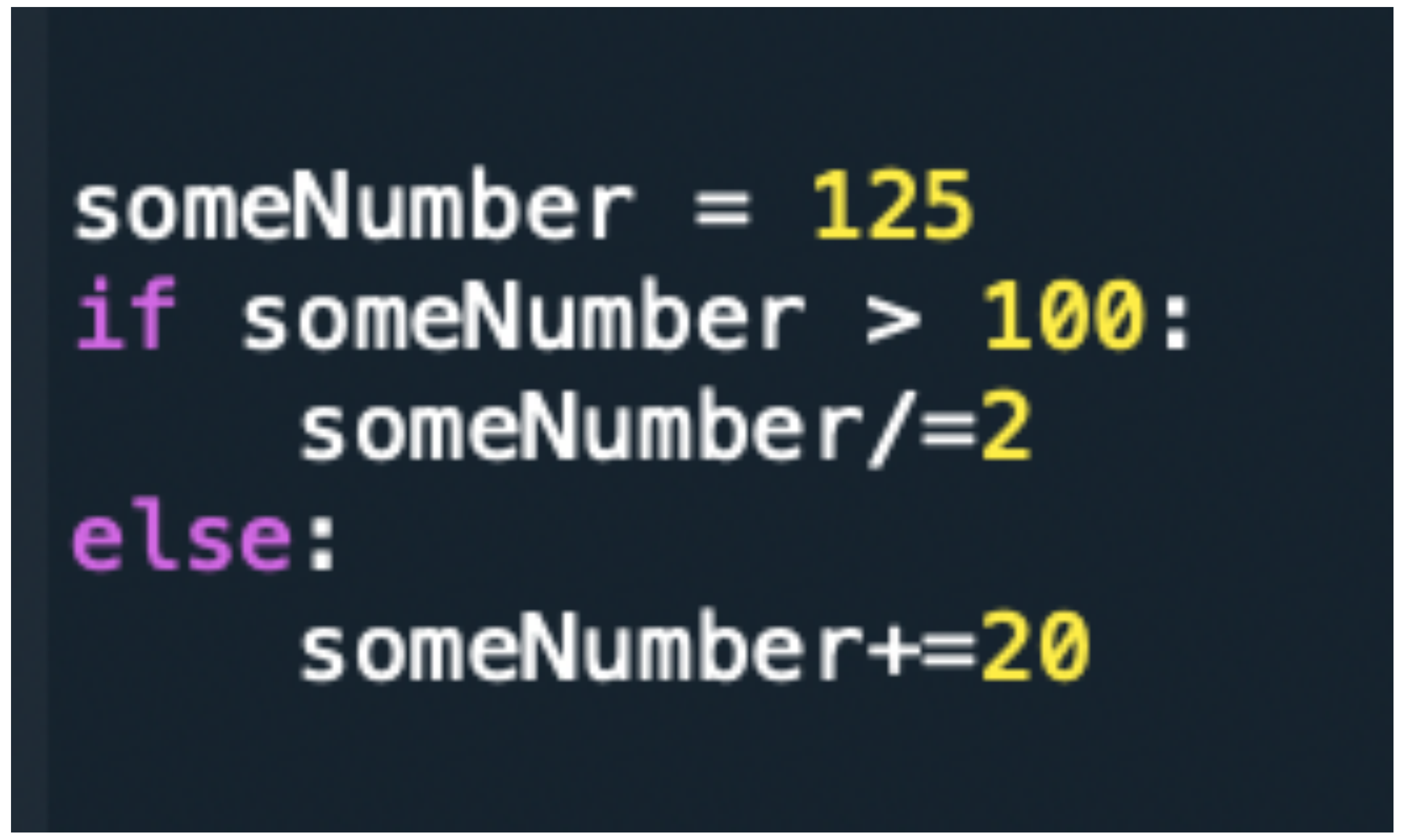
```
"I am Ali".upper()
[1,2,3,4].append(5)
[1,2,3,4,5].count()
```

Functional

Programming paradigms

Structured/Procedural

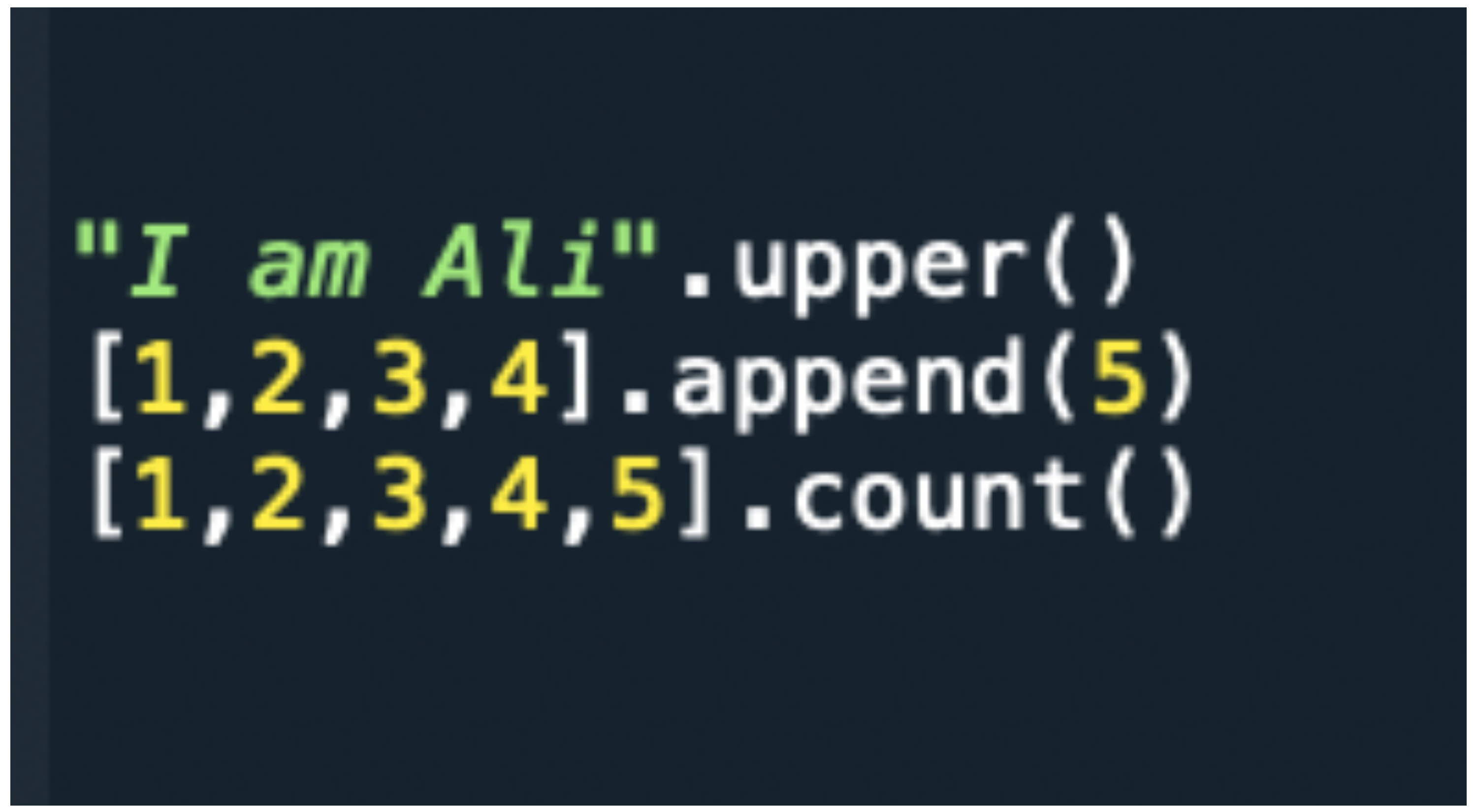
- Hard to read and lengthy
- Hard to reuse (copy/pate)



```
someNumber = 125
if someNumber > 100:
    someNumber/=2
else:
    someNumber+=20
```

OO

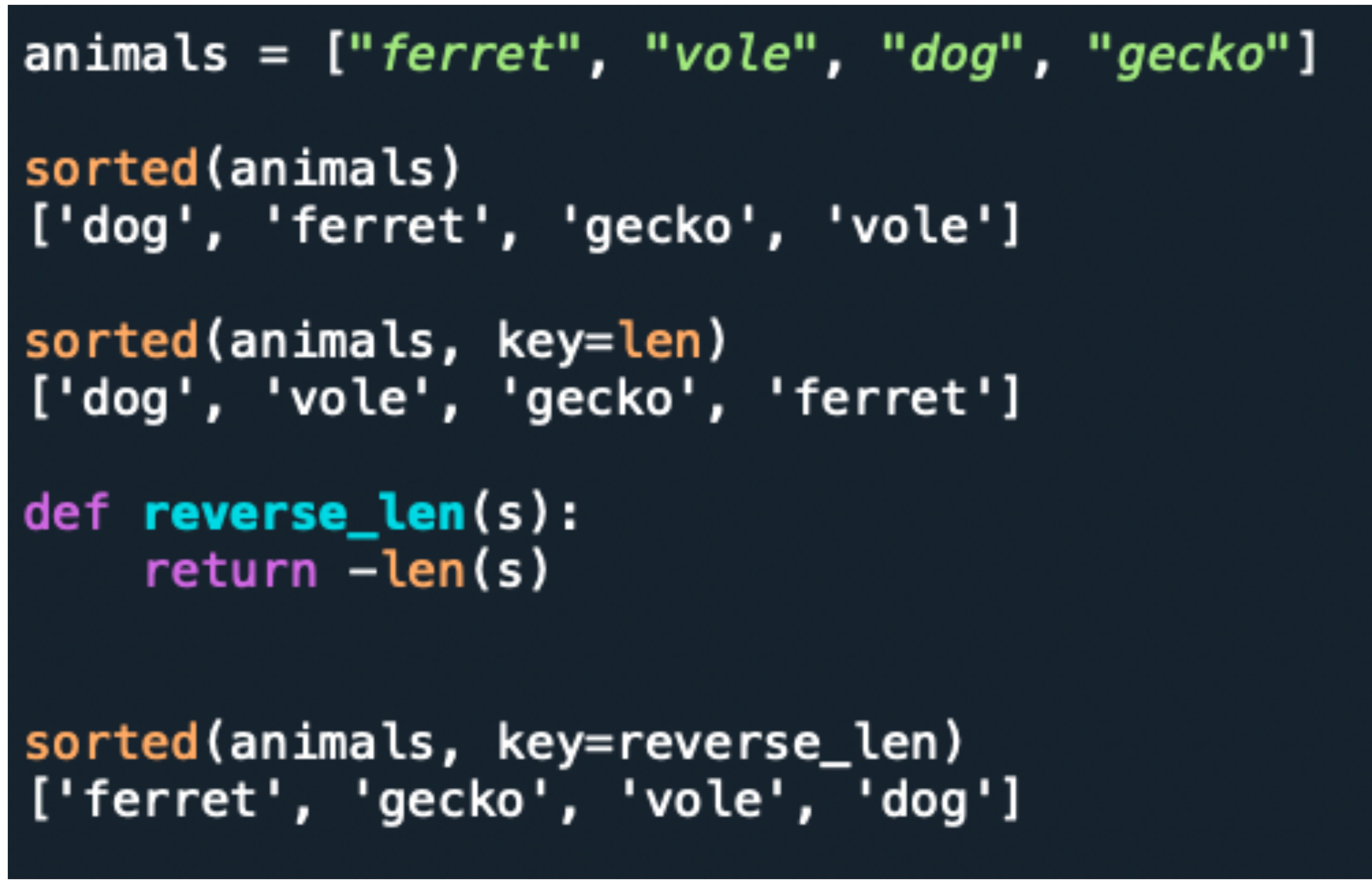
- Smaller chunks (objects)
- If something breaks we go to the object
- Just worry about how objects interact



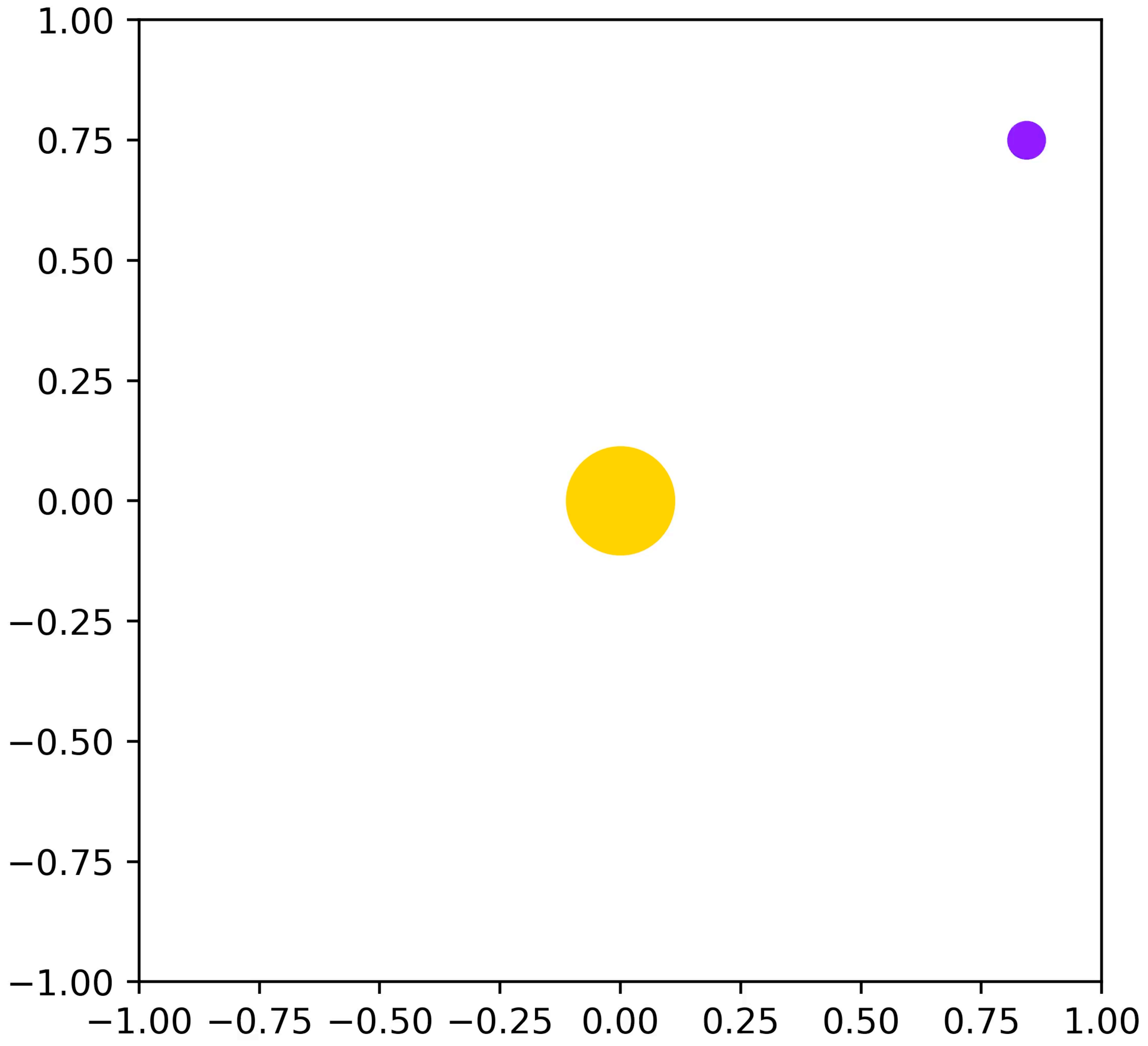
```
"I am Ali".upper()
[1,2,3,4].append(5)
[1,2,3,4,5].count()
```

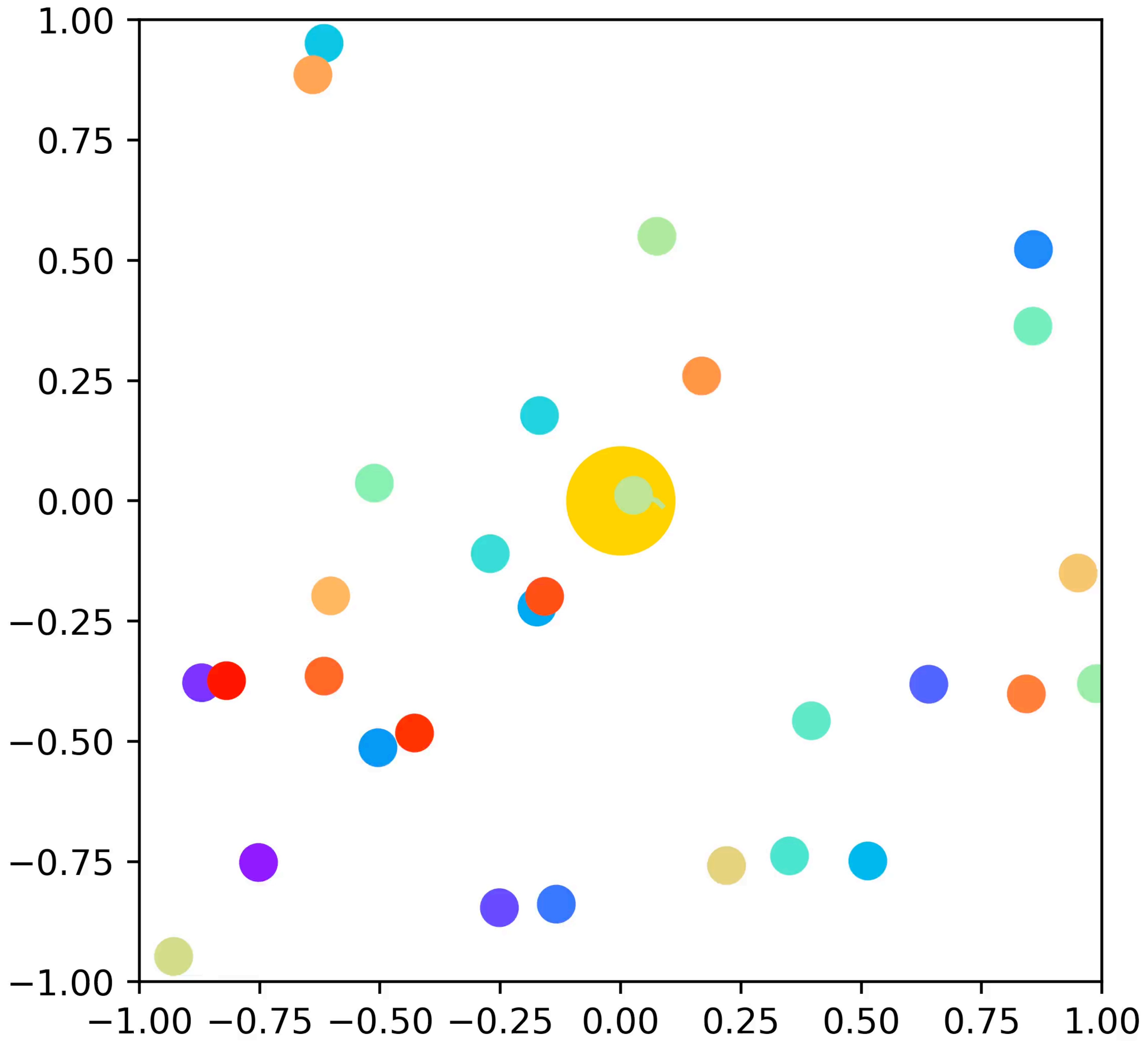
Functional

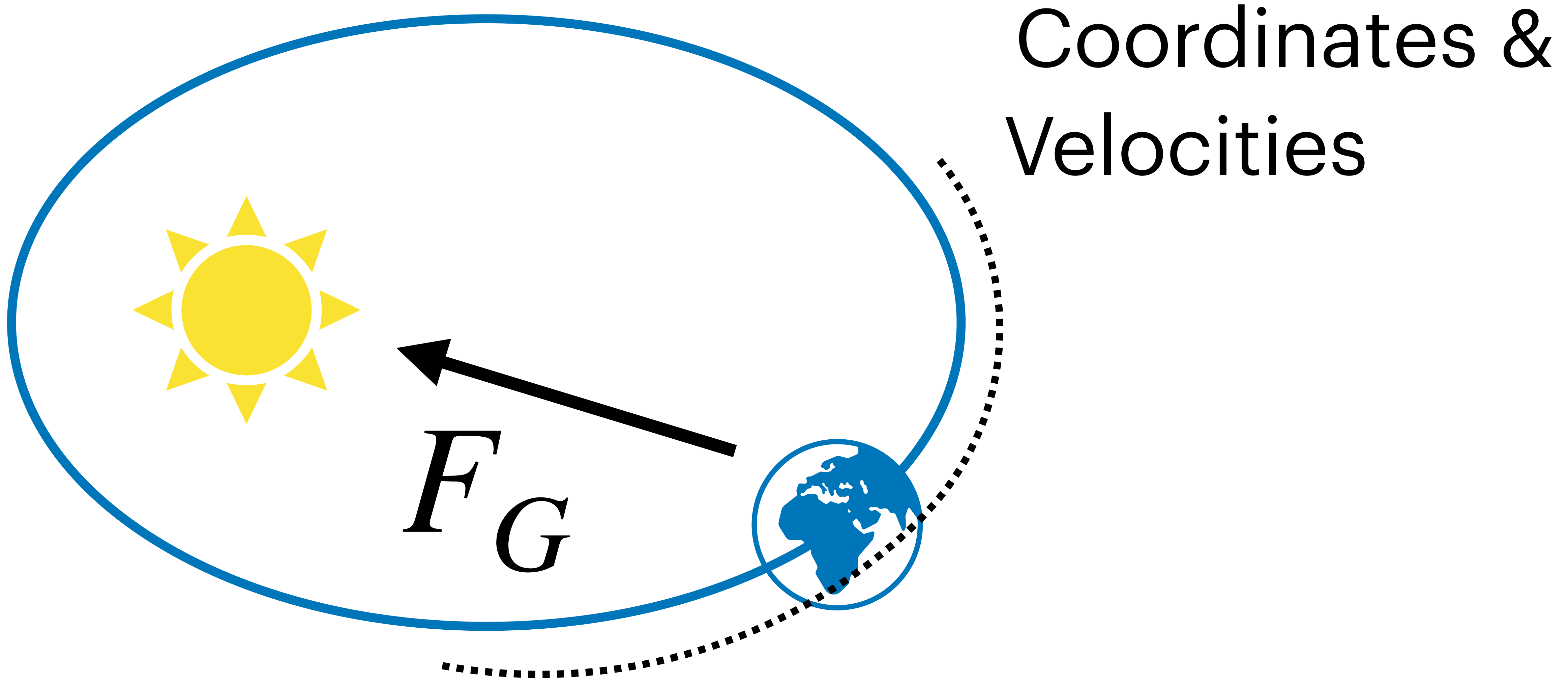
- Describe logic as functions



```
animals = ["ferret", "vole", "dog", "gecko"]

sorted(animals)
['dog', 'ferret', 'gecko', 'vole']

sorted(animals, key=len)
['dog', 'vole', 'gecko', 'ferret']

def reverse_len(s):
    return -len(s)


sorted(animals, key=reverse_len)
['ferret', 'gecko', 'vole', 'dog']
```

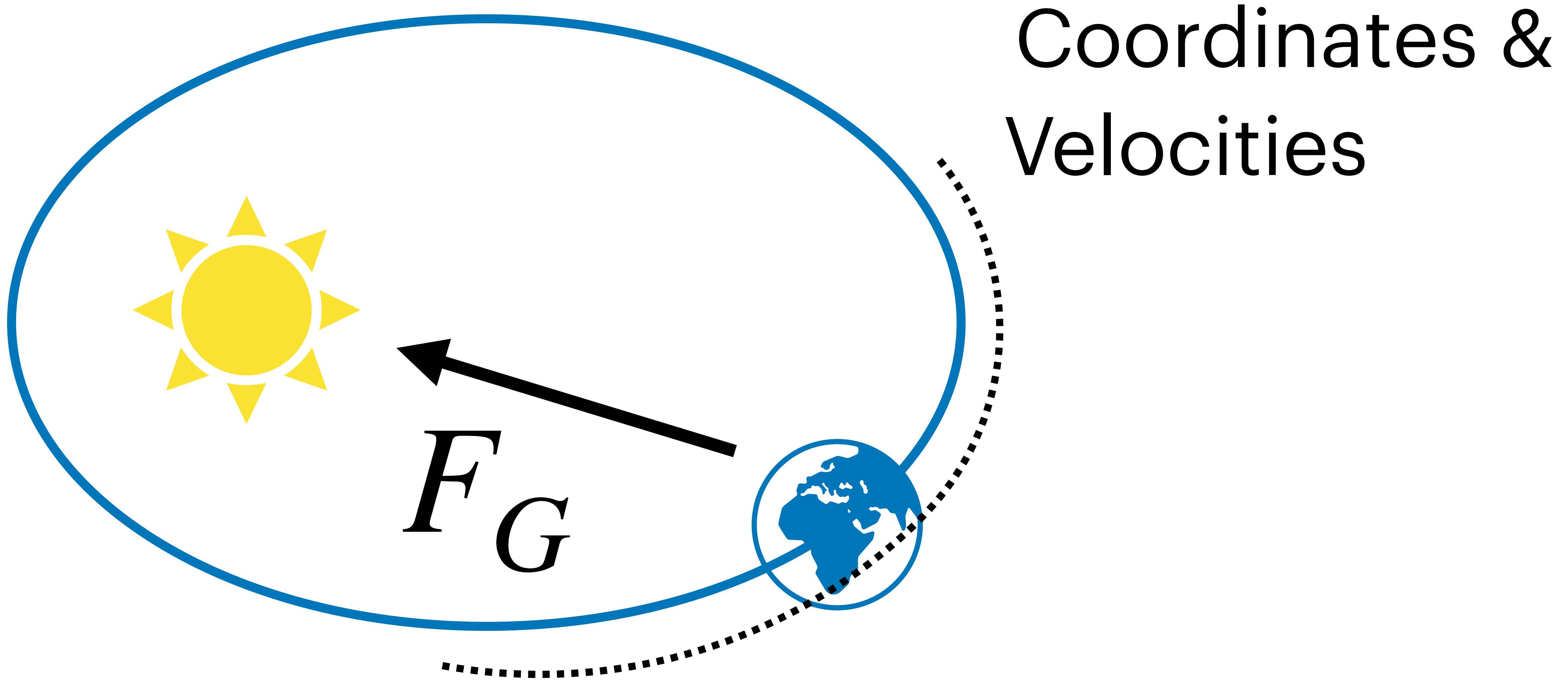

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


Coordinates &
Velocities
F_G

$$\ddot{x}_n = \frac{F_G}{m_{planet}}$$

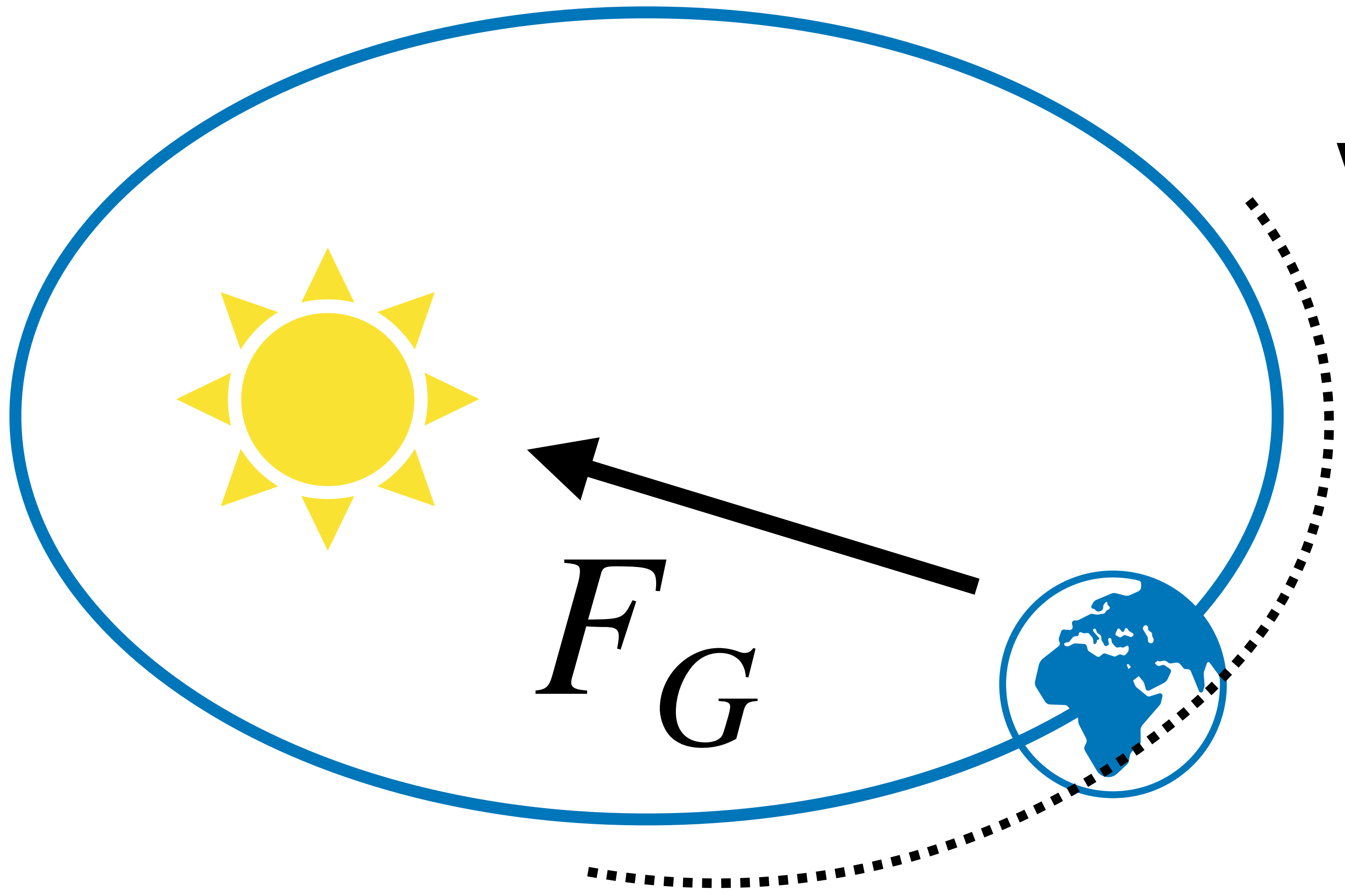
$$F_G = G\frac{M_{Sun} m_{Planet}}{\Delta r^2}$$



$$\ddot{x}_n = \frac{F_G}{m_{planet}}$$

$$F_G = G\frac{M_{Sun} m_{Planet}}{\Delta r^2}$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}} = GM_{Sun}\frac{1}{\Delta r_n^2}$$

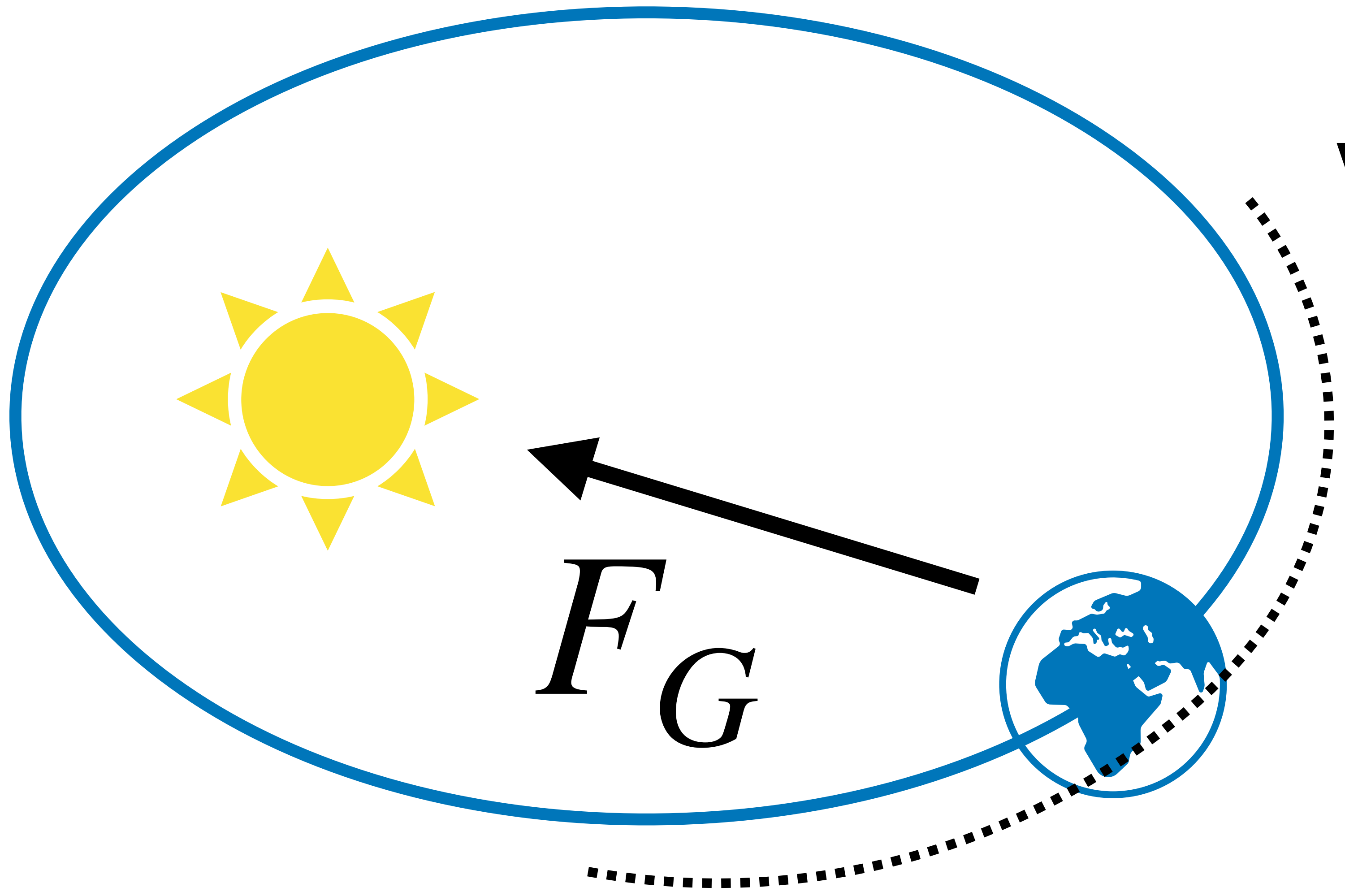


Coordinates &
Velocities

$$\ddot{x}_n = \frac{F_G}{m_{planet}}$$

$$F_G = G\frac{M_{Sun}m_{Planet}}{\Delta r^2}$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}} = GM_{Sun}\frac{1}{\Delta r_n^2}$$



Coordinates & Velocities

Euler

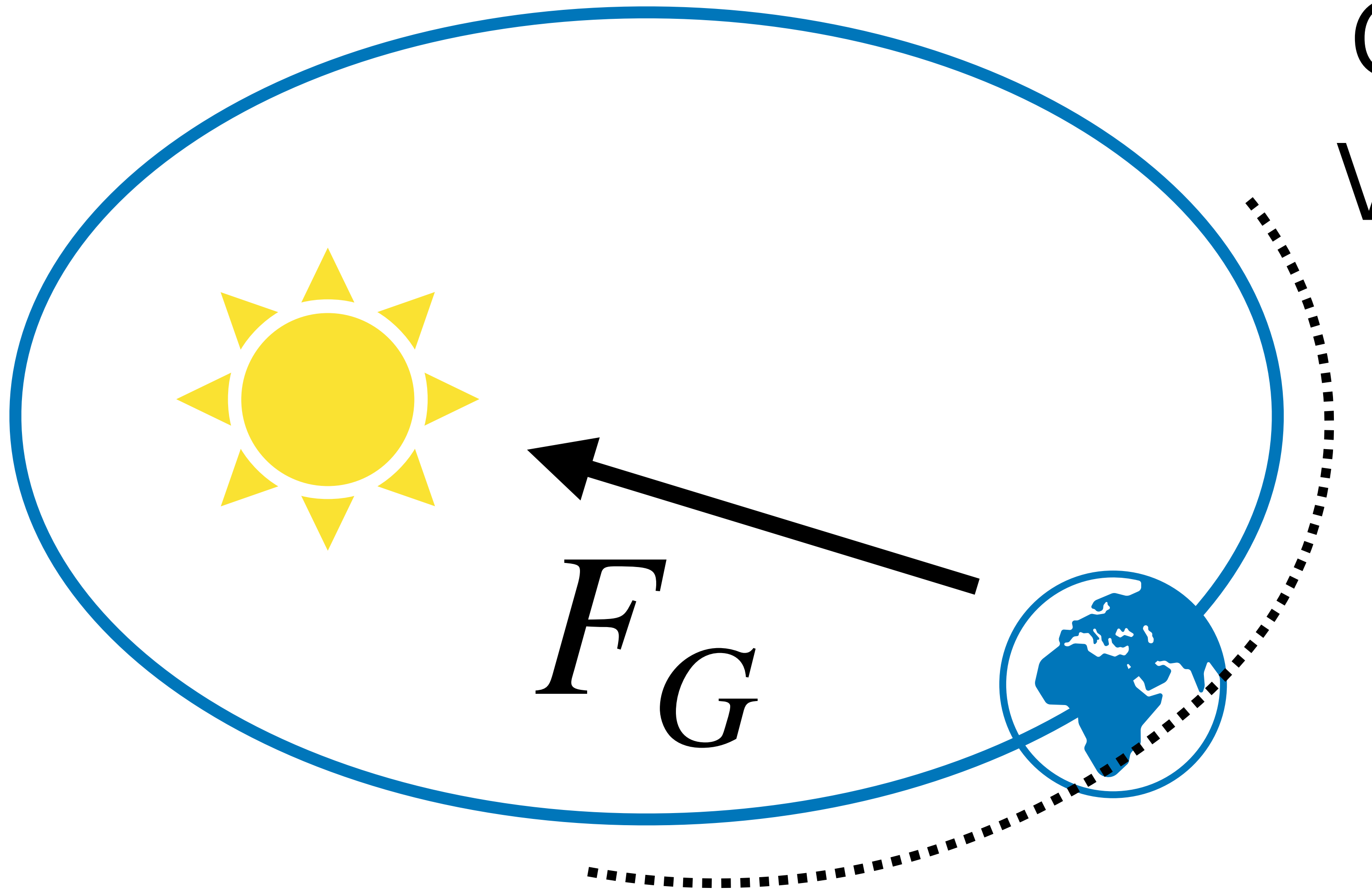
$$x_{n+1} = \frac{1}{2}\ddot{x}_n\Delta t^2 + \dot{x}_n\Delta t + x_n$$

$$\dot{x}_{n+1} = \ddot{x}_n\Delta t + \dot{x}_n$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}}$$

$$F_G = G\frac{M_{Sun}m_{Planet}}{\Delta r^2}$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}} = GM_{Sun}\frac{1}{\Delta r_n^2}$$



Velocity Verlet

$$x_{n+1} = \frac{1}{2}\ddot{x}_n\Delta t^2 + \dot{x}_n\Delta t + x_n$$

Euler

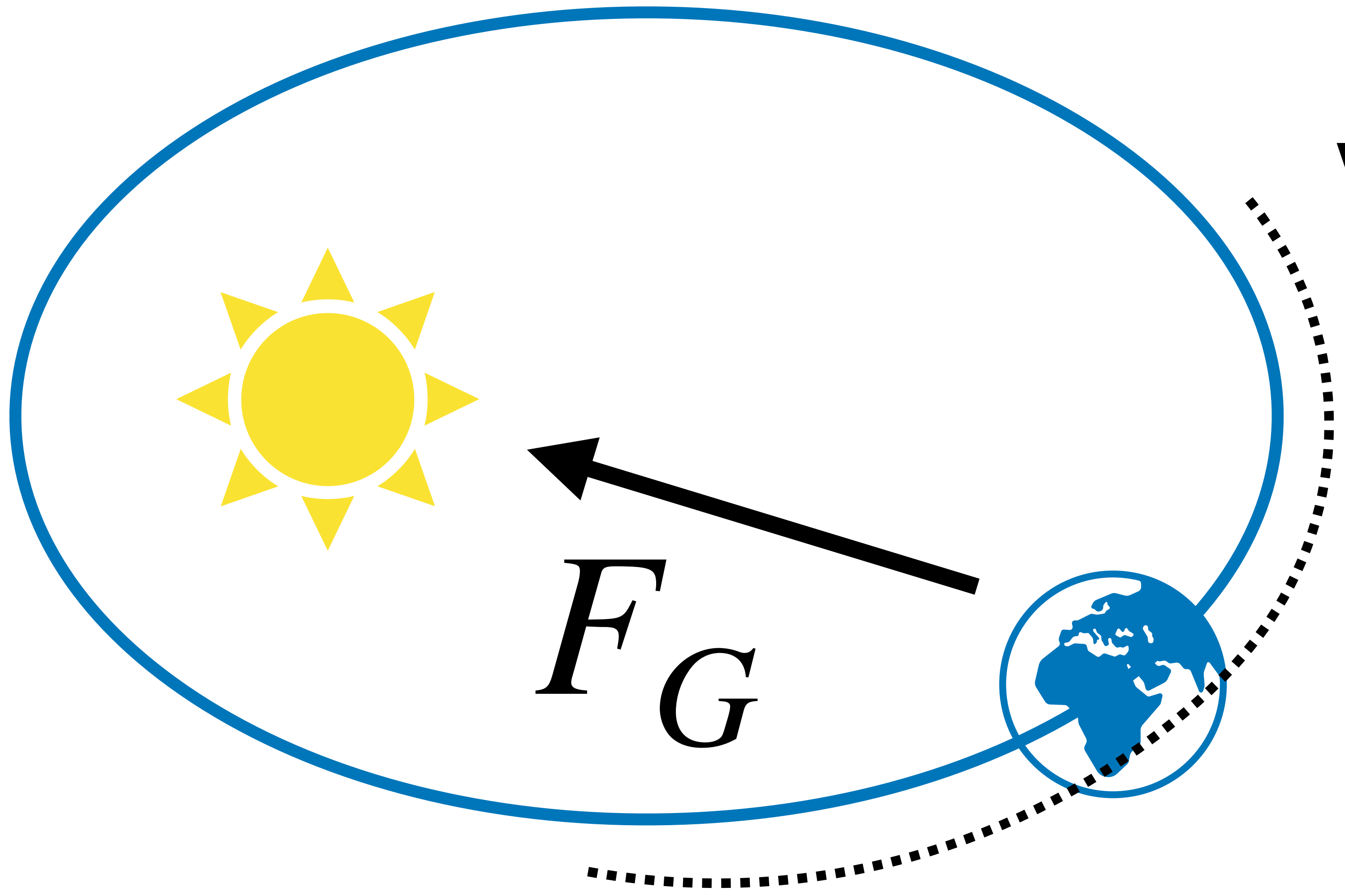
$$x_{n+1} = \frac{1}{2}\ddot{x}_n\Delta t^2 + \dot{x}_n\Delta t + x_n$$

$$\dot{x}_{n+1} = \ddot{x}_n\Delta t + \dot{x}_n$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}}$$

$$F_G = G\frac{M_{Sun}m_{Planet}}{\Delta r^2}$$

$$\ddot{x}_n = \frac{F_G}{m_{planet}} = GM_{Sun}\frac{1}{\Delta r_n^2}$$



Coordinates & Velocities

Euler

$$x_{n+1} = \frac{1}{2}\ddot{x}_n\Delta t^2 + \dot{x}_n\Delta t + x_n$$

$$\dot{x}_{n+1} = \ddot{x}_n\Delta t + \dot{x}_n$$

Velocity Verlet

$$x_{n+1} = \frac{1}{2}\ddot{x}_n\Delta t^2 + \dot{x}_n\Delta t + x_n$$

$$\ddot{x}_{n+1} = GM_{Sun}\frac{1}{\Delta r_{n+1}^2}$$

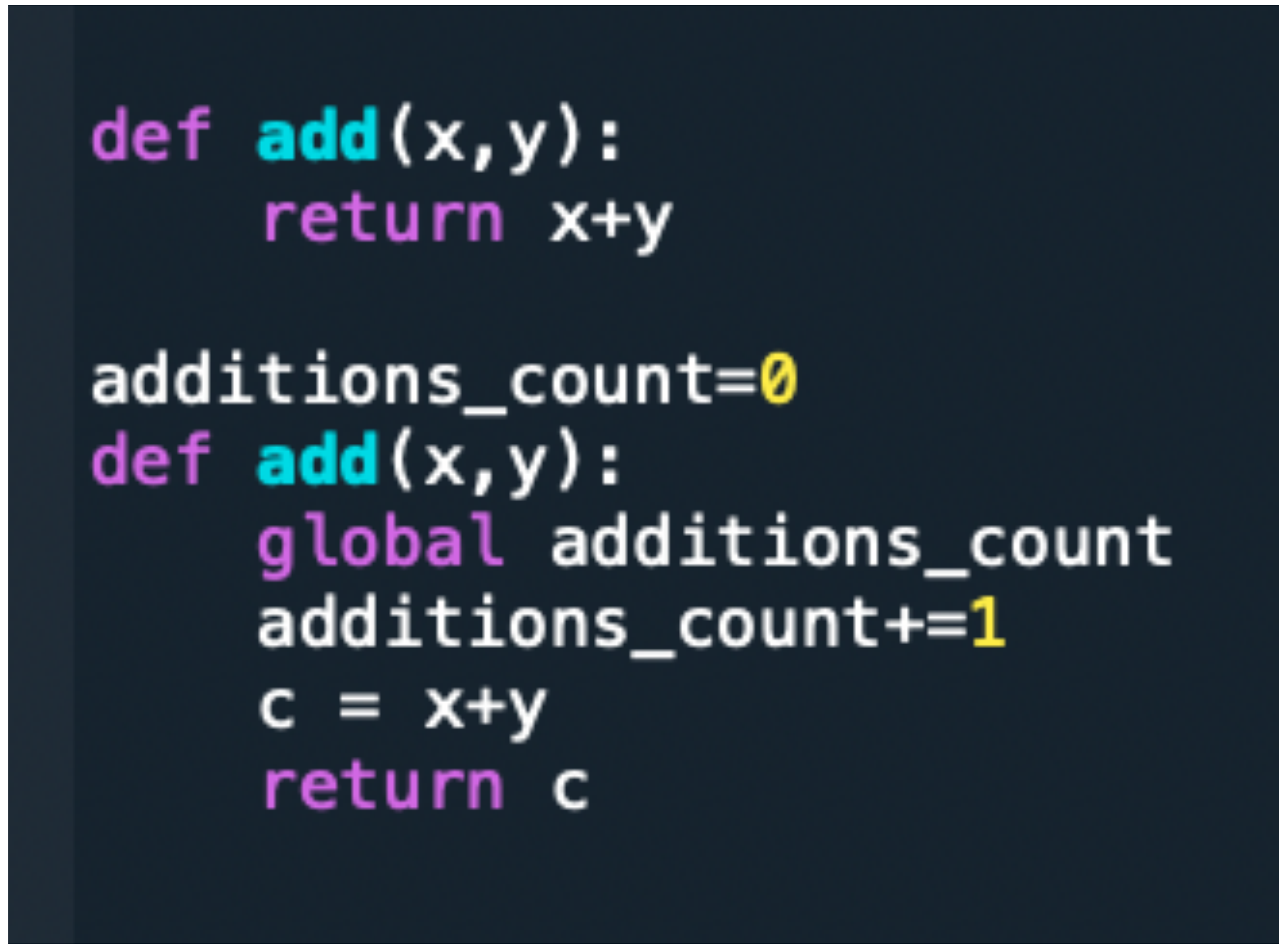
$$\dot{x}_{n+1} = \frac{\ddot{x}_n + \ddot{x}_{n+1}}{2}\Delta t + \dot{x}_n$$

Hands-on
write orbit code

Modules

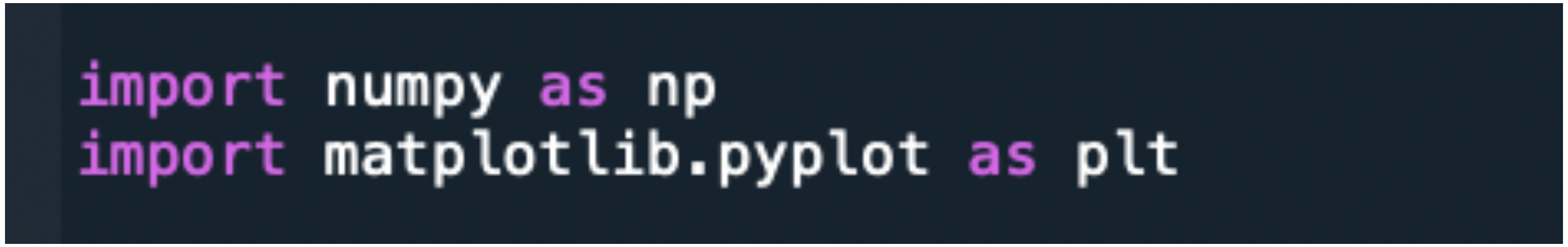
Modular design

- Each module does one job
  - Change code without breaking it
  - Reuse modules elsewhere
- Pure functions



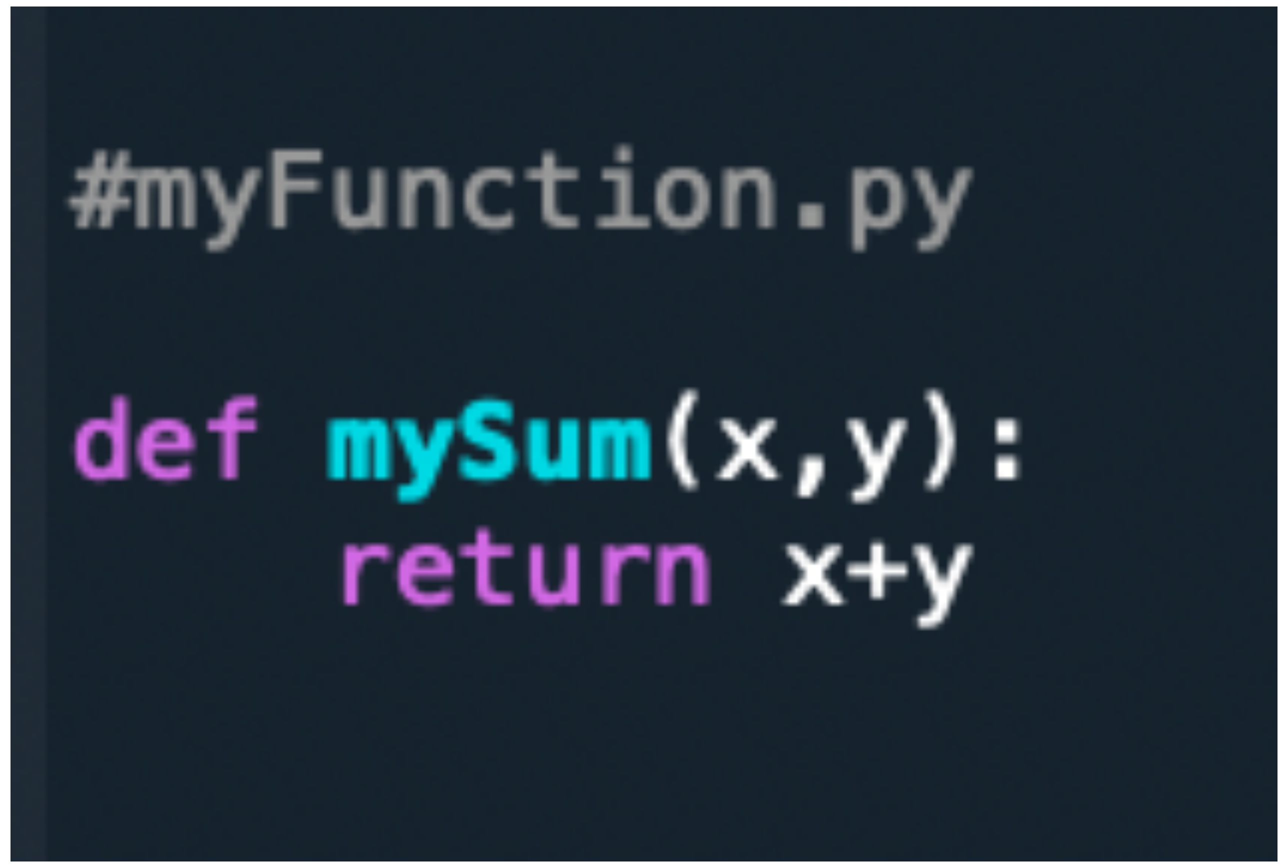
```python
def add(x,y):
    return x+y

additions_count=0
def add(x,y):
    global additions_count
    additions_count+=1
    c = x+y
    return c
```

Namespaces

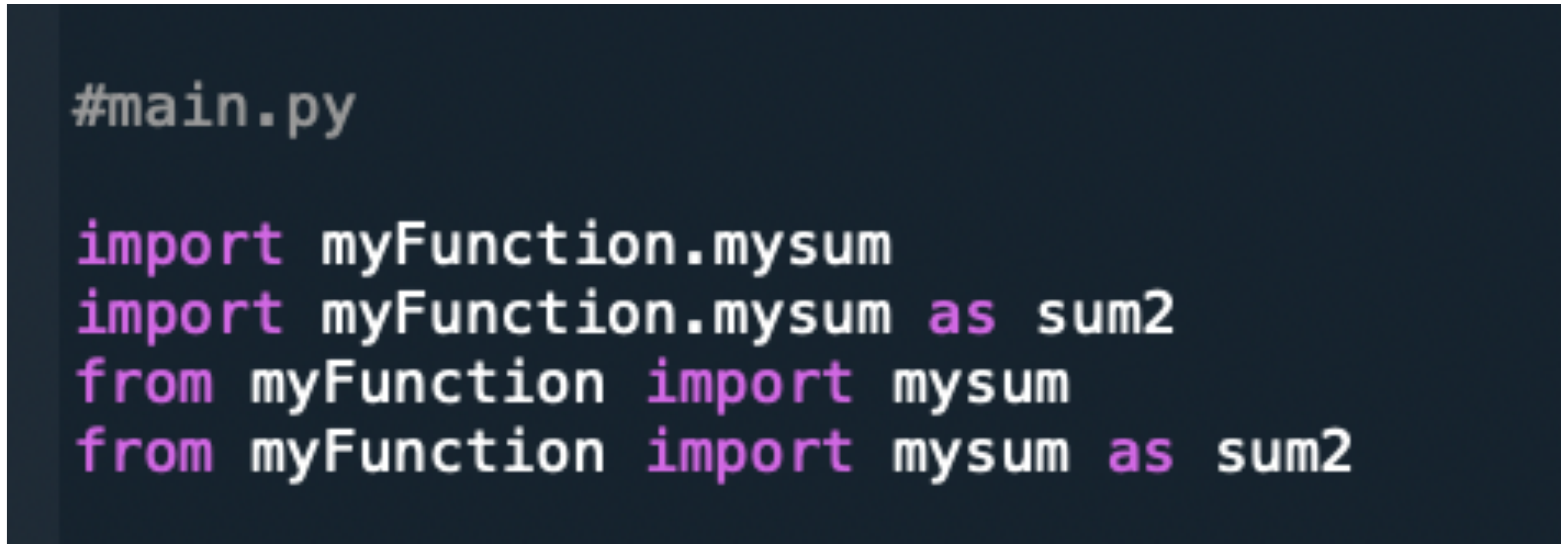


```python
import numpy as np
import matplotlib.pyplot as plt
```

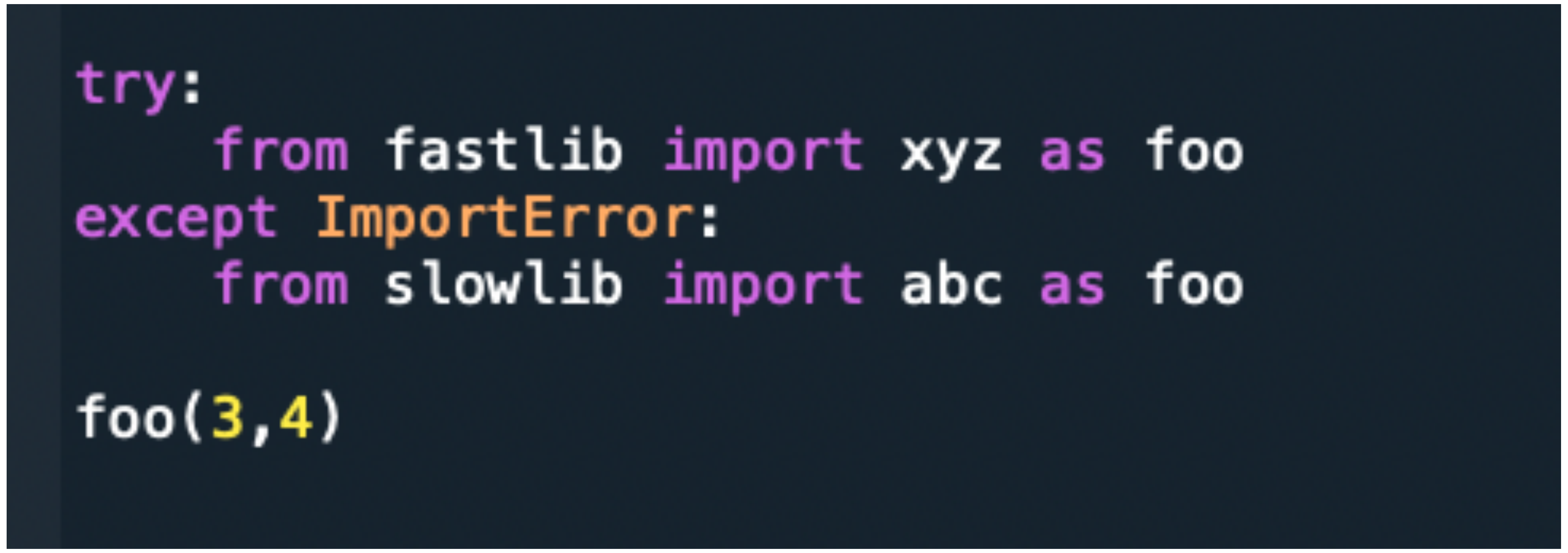
Namespaces



```python
#myFunction.py

def mySum(x,y):
    return x+y
```



```python
#main.py

import myFunction.mysum
import myFunction.mysum as sum2
from myFunction import mysum
from myFunction import mysum as sum2
```



```python
try:
    from fastlib import xyz as foo
except ImportError:
    from slowlib import abc as foo

foo(3,4)
```

Hands-on
write orbit code - Modular

Hands-on
write orbit code - Modular
Extend to N planets

$\times N$

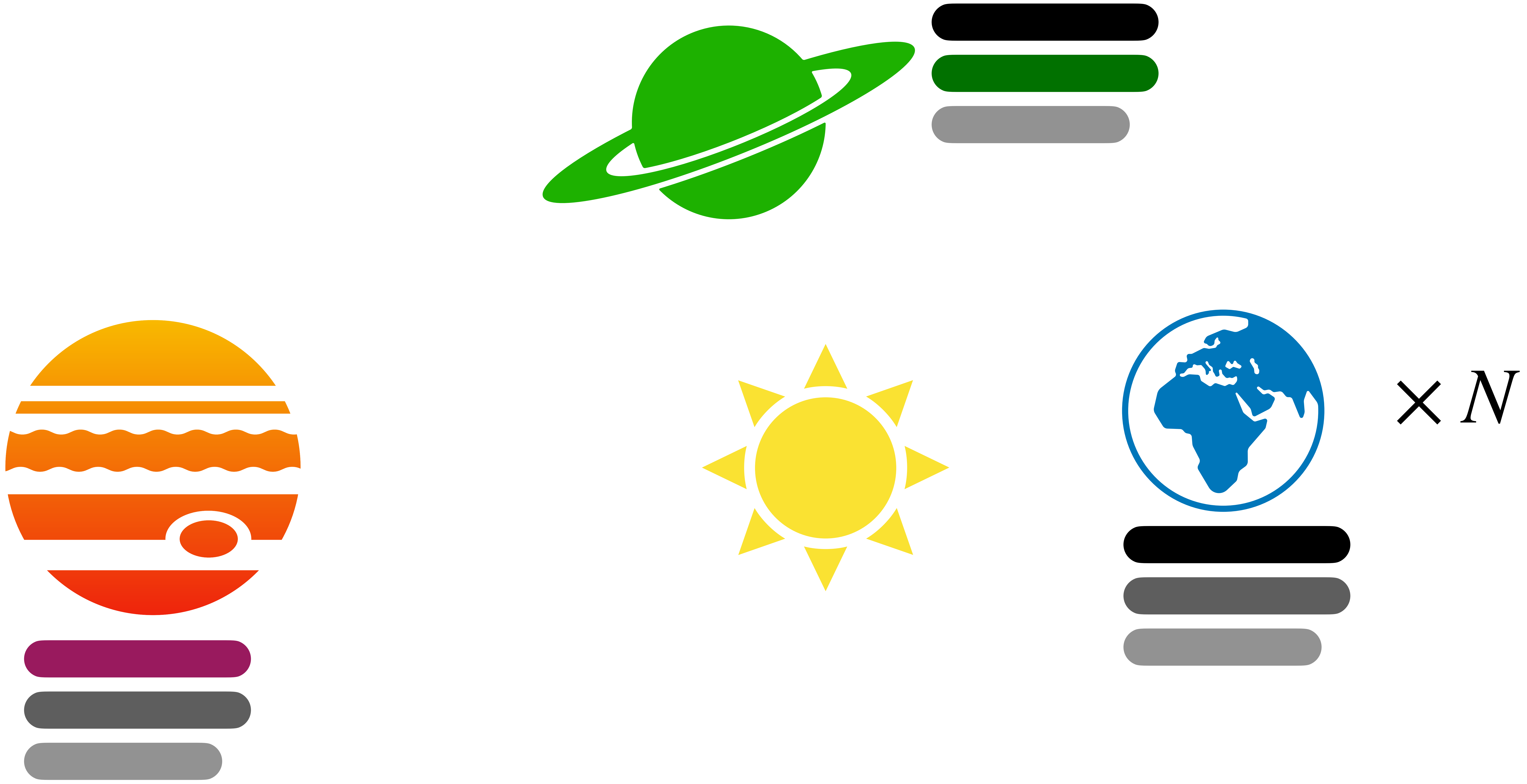
Object Oriented

OO
Goal: Manage complexity

- **Software**
  - Maintenance
  - Evolution
  - Preservation
- **Development**
  - Large projects
- **Flexible software**
  - Work at any level of abstraction

Hands-on
write orbit code - OO

Hands-on
write orbit code - OO
Extend to N planets


$\times N$

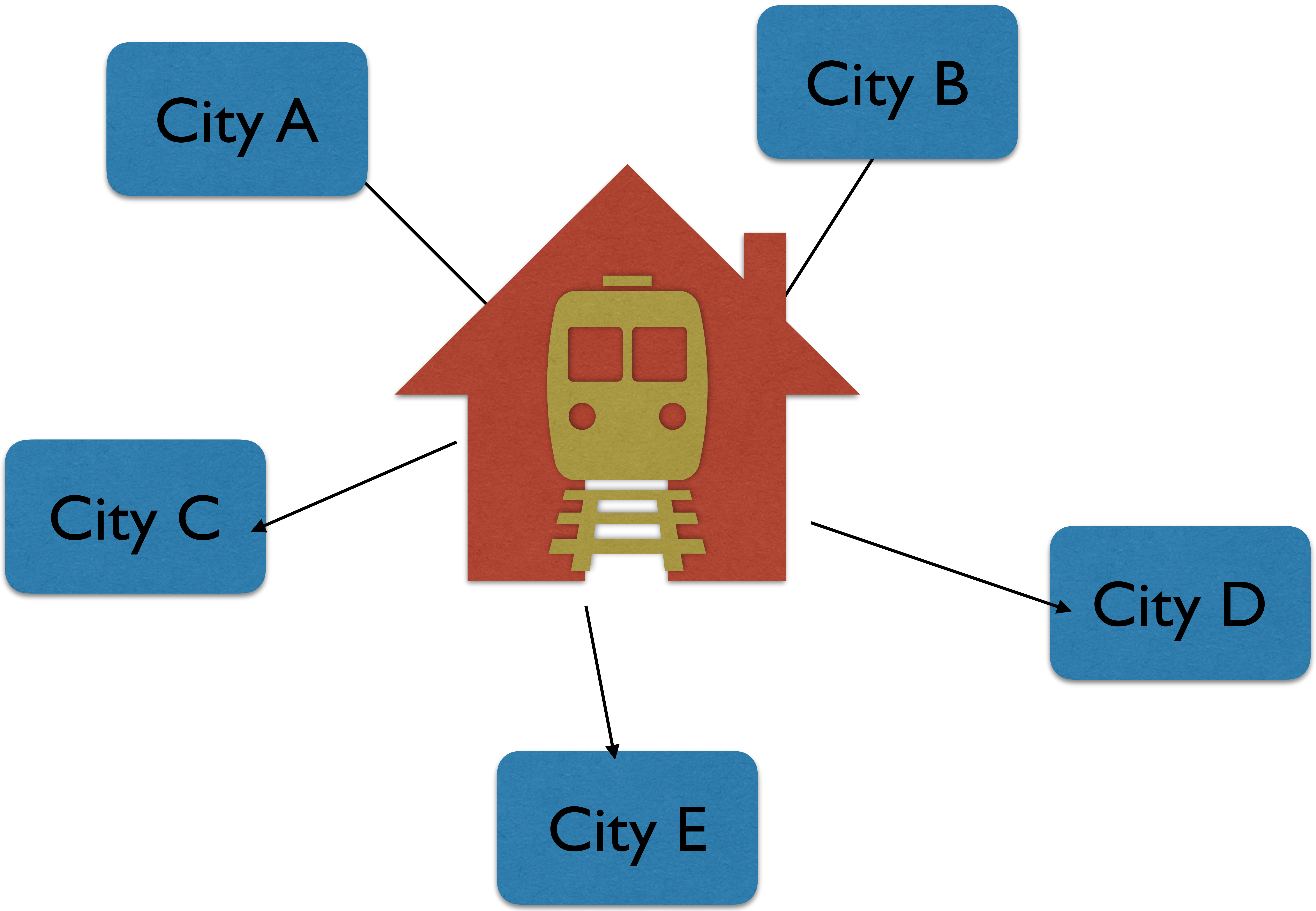
Hands-on
write orbit code - OO
Extend to N planets and more planets

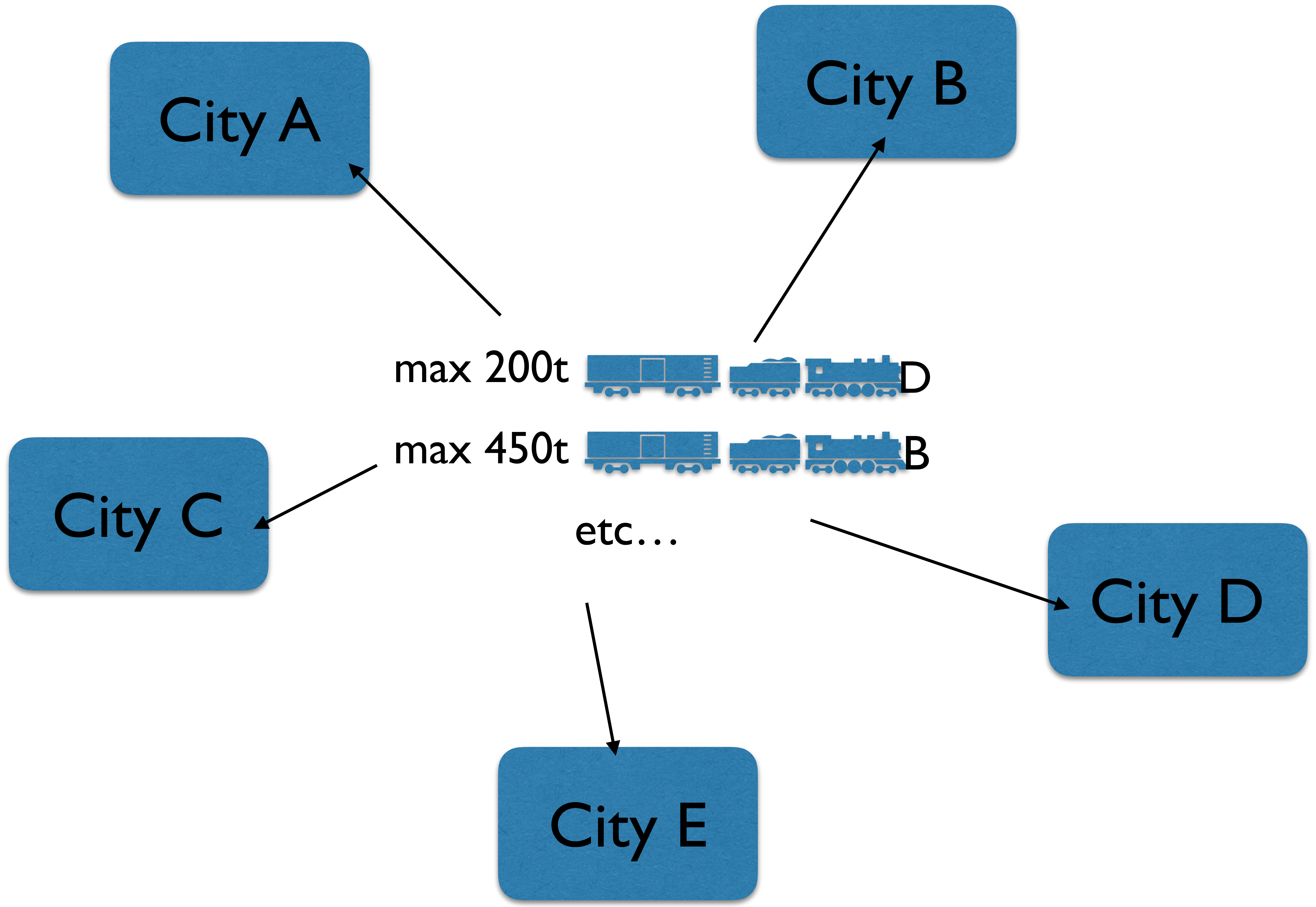
Hands-on
write orbit code - OO

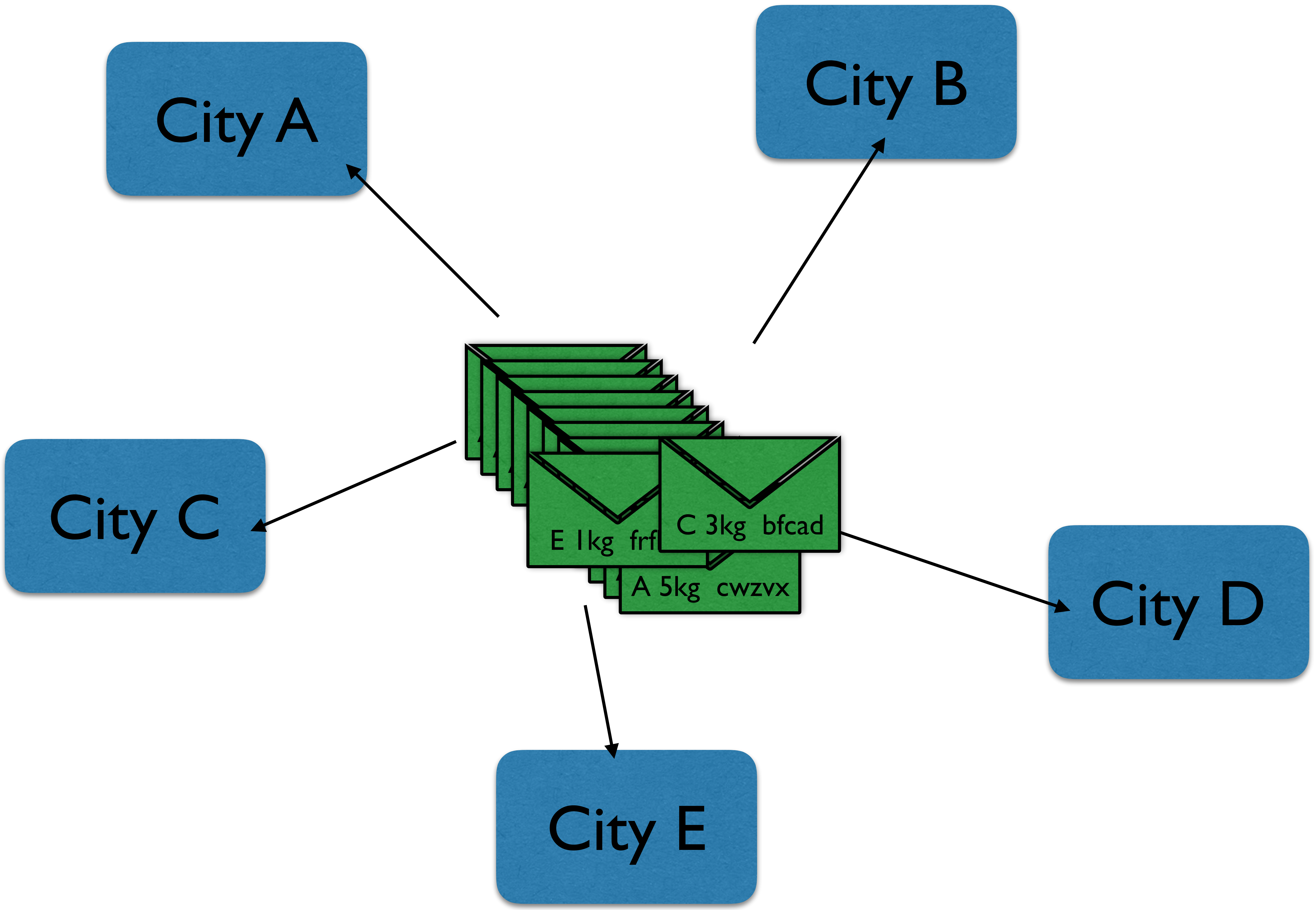
Class - Overloading

Operator	Expression	Internally
The string representation	str	__str__(self)
The number of elements	len	__len__(self)
Check membership	in	__contains__(self, value)
Index operator	[index]	__getitem__(self, index)
Addition	+	__add__(self, value)
Subtraction	-	__sub__(self, value)
Multiplication	*	__mul__(self, value)
Power	**	__pow__(self, value)
Equal to	==	__eq__(self, value)
Greater than	>	__gt__(self, value)
Bitwise Right Shift	>>	__rshift__(self, value)
Bitwise NOT	~	__invert__(self)

Exercise: a freight station



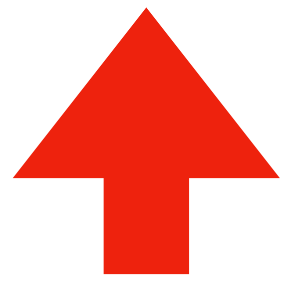

City A
City B
City C
City D
City E
max 200t
D
max 450t
B
etc…


City A
City B
City C
City D
City E
E 1kg frf
C 3kg bfcad
A 5kg cwzvx

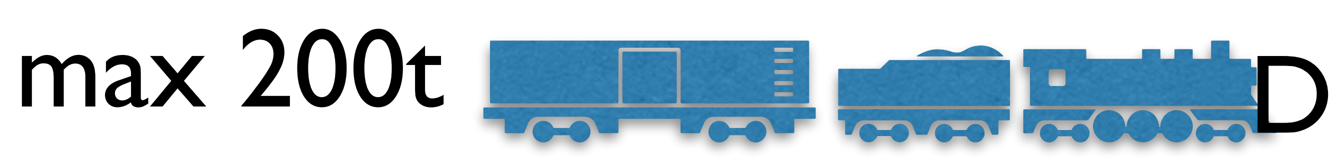
Design an OO model for the station

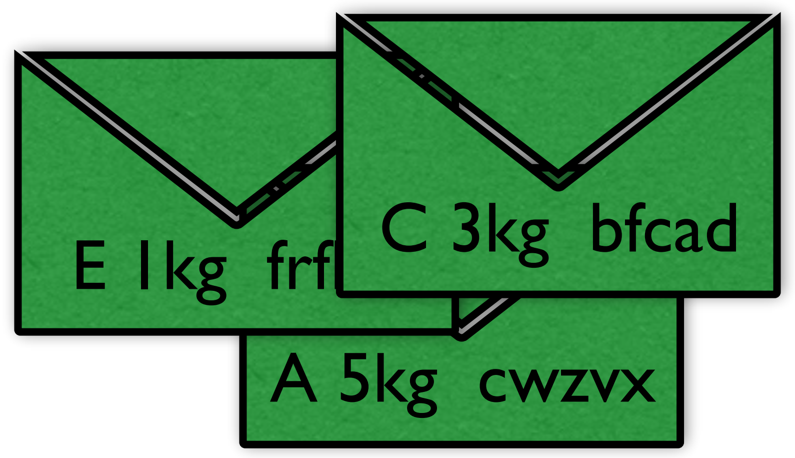
classes, objects, interfaces, public/private, which methods/state

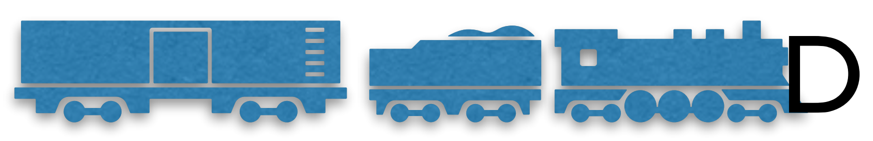
but no implementation!



a random train arrives, is loaded with correct mail, leaves, and repeat

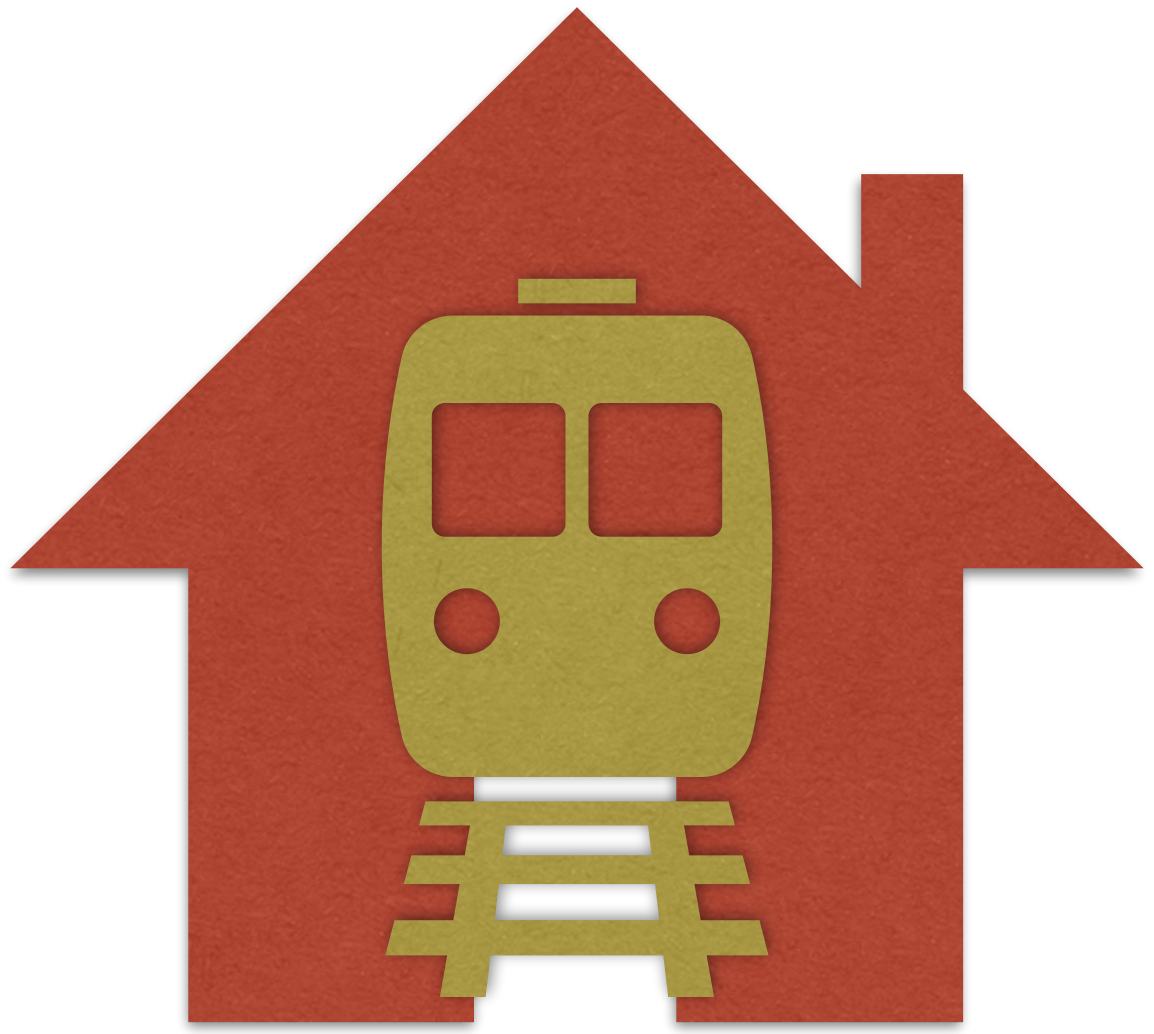






Design an OO model for the station

classes, objects, interfaces, public/private, which methods/state

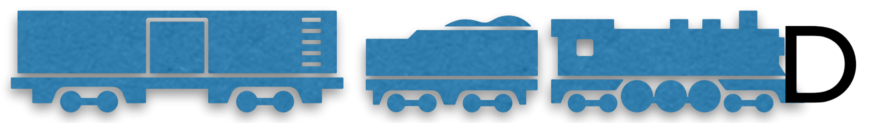


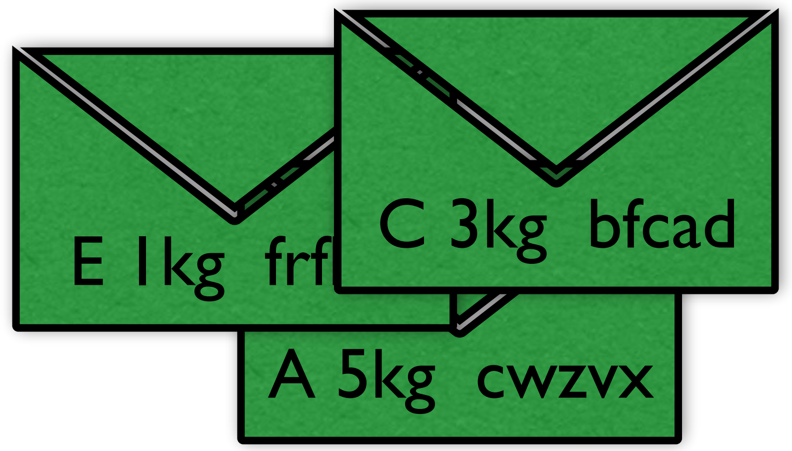
Write the props of each train to a separate file named train###.txt

Train: 001
Destination: D
Capacity: 140000
Actual load: 139999.31
Number of parcels: 40015
Parcels:
  D 3.125
  D 3.817

a random train arrives, is loaded with correct mail, leaves, and repeat

max 200t




E 1kg frf
C 3kg bfcad
A 5kg cwzvx