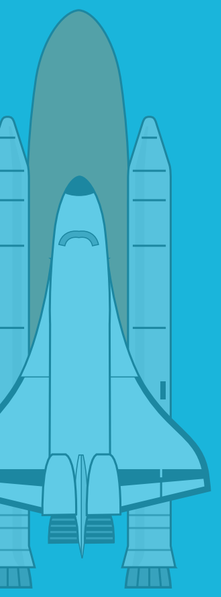
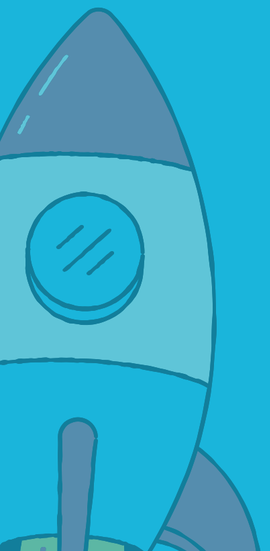
Creating & Using custom Classes

MASTER PYTHON CLASSES





PROBLEM SOLVED BY CLASSES

1 UNDERSTANDING CLASSES

- Classes are the building blocks of Python script
- Instance of Class is called an Object. Instance is copy of the Class.
- Inbuilt Data Types are also implemented as Classes

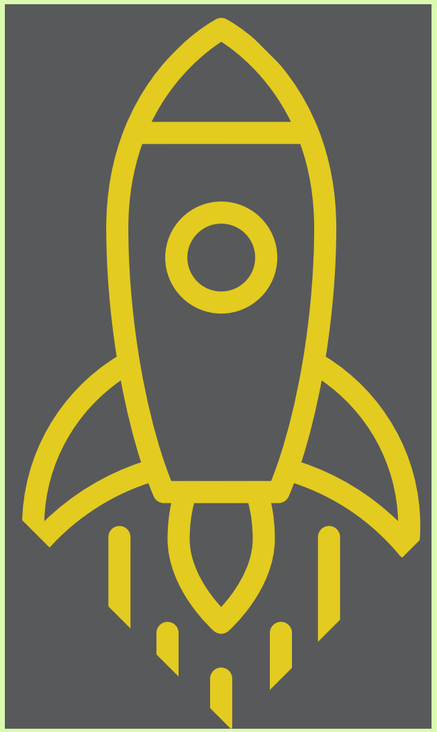
2 WHAT IS THE PROBLEM?

Each application will have different kind of variables which have different set of functions. Need to Implement them, and share them around

3 HOW CLASS SOLVES IT

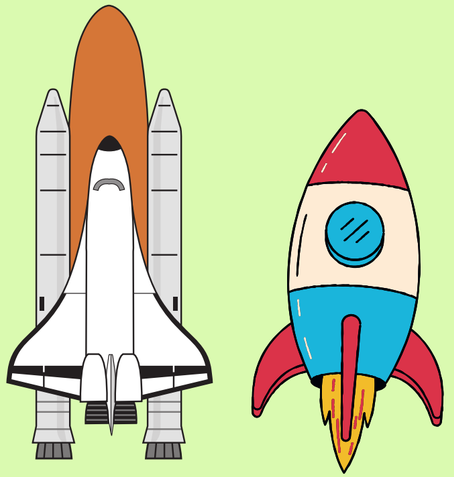
- Enables the Architect to develop the templates
- The templates are used to create Objects as copy of Classes
- Top Down / Bottom Up design can be implemented

CLASS IN ACTION



Rocket requires:
Name, Height, Weight,Diameter

Rocket can:
Launch, Rotate, Turn



2 Instances Solum, Prime

rocket1 = Rocket('solum', 250,7807,325)
rocket2 = Rocket('prime', 157,5267,170)

Rocket class:

1. Uses python to work in Rocket Science application
2. Implements the template based on the Rocket size and Capability
3. The Capabilities are implemented as methods, which provide data and allow to manipulate the instance

CLASS ROCKET

Idea : Creating the Rocket Template (Class)

```
class Rocket:
    def __init__(self,name,height,weight,diameter):
        self.name = name
        self.height = height
        self.weight = weight
        self.diameter = diameter
        self.load = None

    def launch(self,altitude):
        position = self.height + altitude
        return f'Position of {self.name} is {position}'

    def rotate(self,angle):
        if angle > 90:
            print(f'Rocket {self.name} is headed to Moon')
        elif angle > 180:
            print(f'Rocket {self.name} is headed to Earth')
        else:
            print('I see, the target is sun')
```

CODE EXPLANATION

- Class Rocket is declared
- Rocket's init function is the starting point for Object / Instance
- launch and rotate method do rocket science calculation. They are functions

You have to understand only this level, and you can make rockets... or anything you want "Inside Python"

INTERACTION METHODS

Idea : Two instances interacting with each other

```
def take_load(self, other):
    self.load = other.name
    print(f'{self.name} has taken a load!!!')

def have_load(self):
    if self.load:
        print(f'Yes. Load is {self.load}')
    else:
        print(f'No Load. {self.name} is Free')
```

CODE EXPLANATION

- Instances or the copies of classes can be programmed to interact and provide output that is different from the usual.
- Using the '.' operator along with the instance, is the way of passing argument to 'self'
- This is how the complexity of the classes is hidden, and refactored. If you want, now you can look under the hood

MAKING COPIES INTERACT

Idea : Two different rockets from one Class

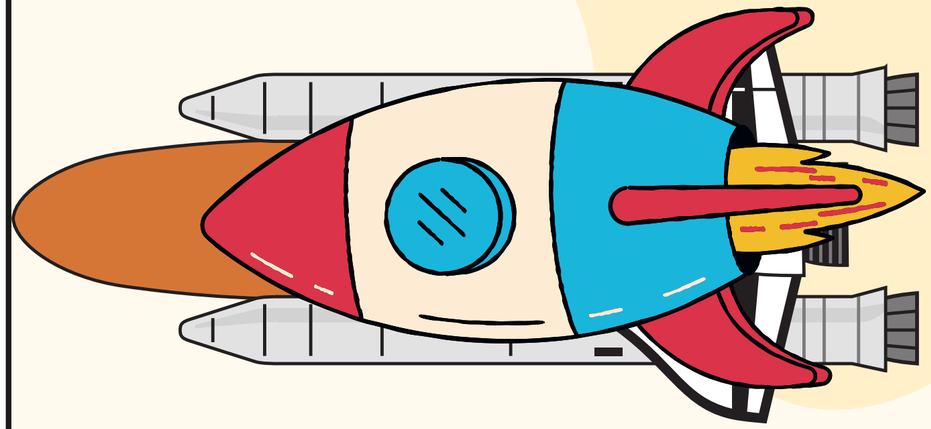
rocket1 = Rocket(self,'solum',250,7807,325)

rocket2 = Rocket(self,'prime',157,5267,170)

Idea : Making instances to interact using methods

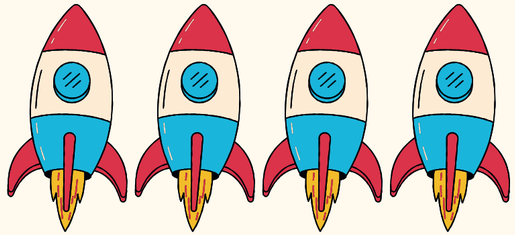
rocket1.takeload(rocket2)

rocket1.hasload() => Yes, Load is Prime



Prime on top of Solum

List of Rockets =



The objects created can be used in data structures like list, dict. They will work, and allow flexibility in using the objects

BEFORE CLASSES?? MODULES

Idea : functions in different file can be imported into your script

my_func.py:

```
mod_name = 'This is nice'

def print_name():
    print('There is a module.')
```

your_script.py:

```
import my_func
my_func.print_name() => This is a module
print(my_func.mod_name) => This is nice
```

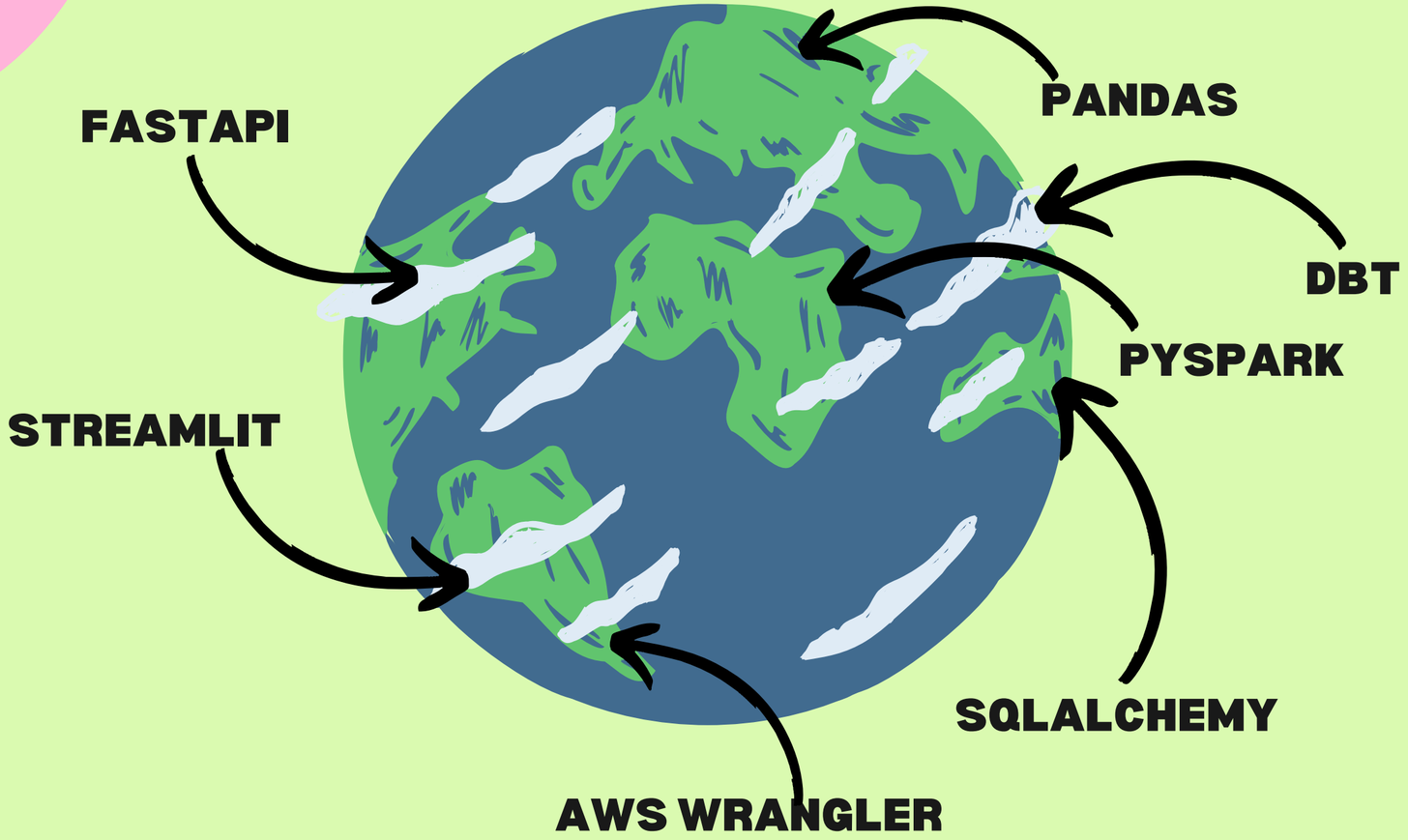
MODULES ARE NOT TEMPLATES

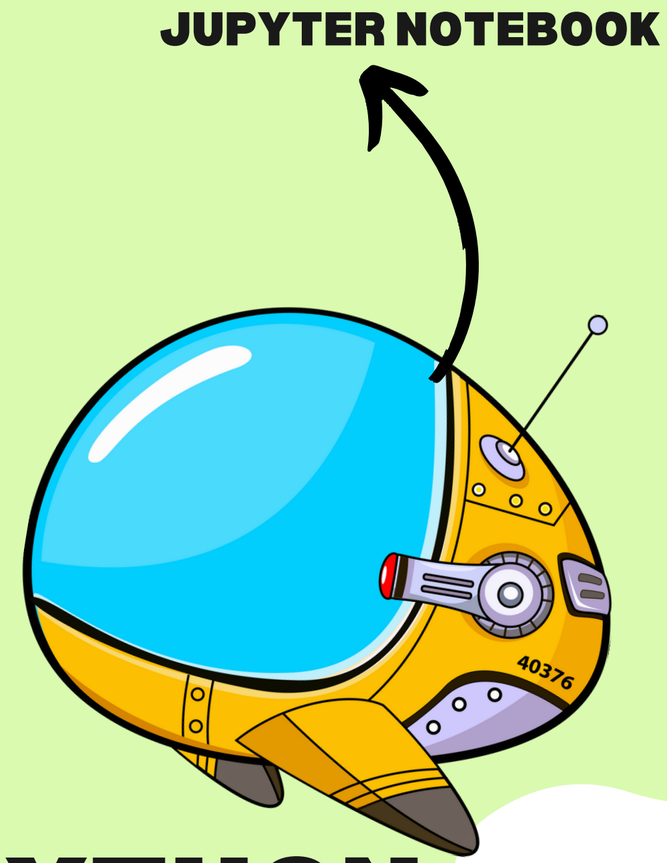
- **Disadvantage** of modules is we cannot create many copies using it.
- We cannot use it as variable inside another data structure
- **Advantage** is module allows for refactoring complex script.
- Makes packaging and sharing the code easy.

Read a lot of Code!!!!

You have taken the first steps into Python Ecosystem.

We will confidently work with libraries, new classes, create charts and make wonders with python





BLAST OFF TO THE PLANET PYTHON