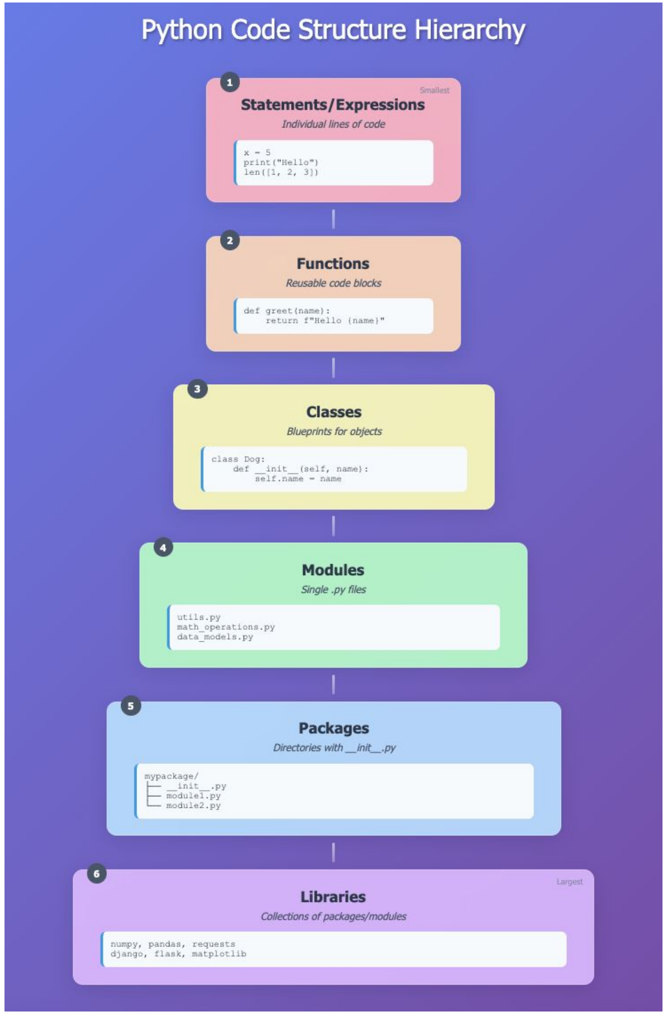
Recall - Week 3

You should be able to:

- Apply conditional statements to initiate decisions using Python code
- Incorporate conditional statements and control structures within Python functions.
- Distinguish different forms of for-loops and while-loops, including the use of enumerate and zip().
- With a basic knowledge of Python lists, begin looping over sequences of data



Python Code Structure Hierarchy

1 Smallest

Statements/Expressions

Individual lines of code

```
x = 5
print("Hello")
len([1, 2, 3])
```

2

Functions

Reusable code blocks

```
def greet(name):
    return f"Hello {name}"
```

3

Classes

Blueprints for objects

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

4

Modules

Single .py files

```
utils.py
math_operations.py
data_models.py
```

5

Packages

Directories with __init__.py

```
mypackage/
├── __init__.py
├── module1.py
└── module2.py
```

6 Largest

Libraries

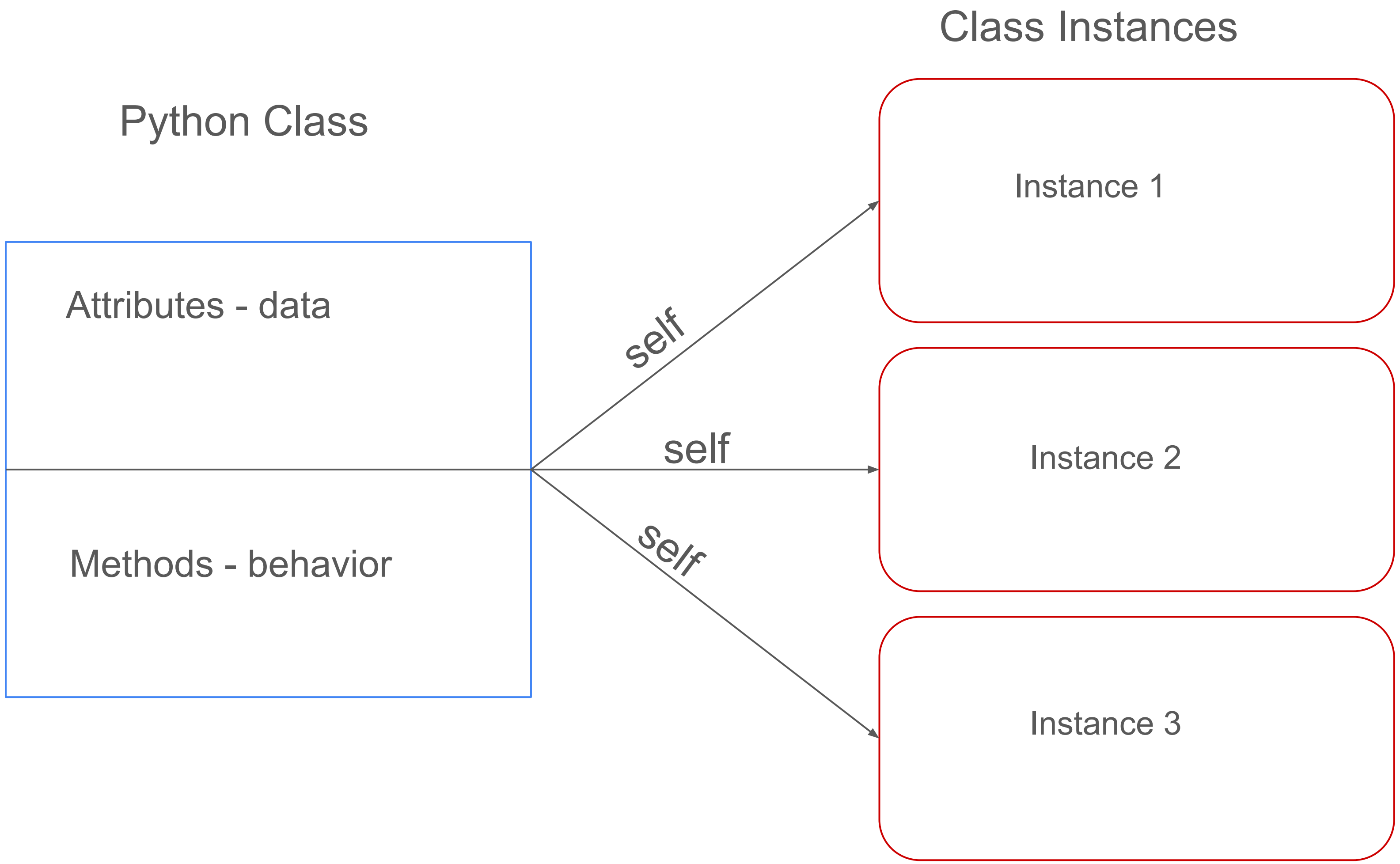
Collections of packages/modules

```
numpy, pandas, requests
django, flask, matplotlib
```

Week 4

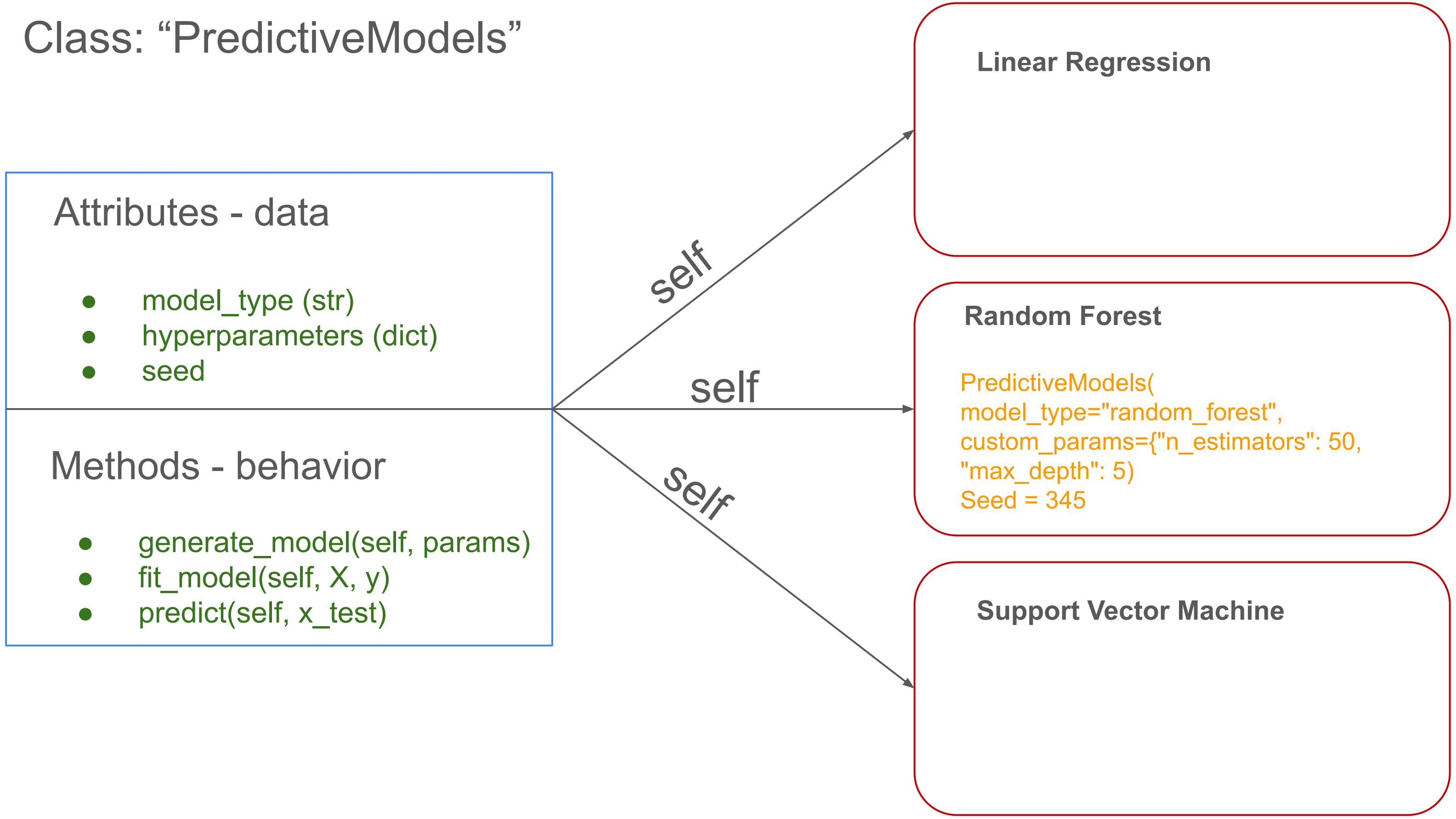
You should be able to:

- Define and instantiate a Python class
- Create attributes and methods (class & instance)
- Access attributes and methods through the class or instance
- Exploit class inheritance to streamline your code


Class Instances
Python Class
Attributes - data
Methods - behavior
self
self
self
Instance 1
Instance 2
Instance 3

Instances

Class: “PredictiveModels”



```
PredictiveModels(
model_type="random_forest",
custom_params={"n_estimators": 50,
"max_depth": 5)
Seed = 345
```

class PredictiveModels:

```
# Initialize instance attributes: Data

    def __init__(self, model_type, parameters):

        self.model_type: str = model_type
        self.parameters = parameters
```

```
class PredictiveModels:


# Initialize Class attributes: Data

          SUPPORTED_MODELS  = {"linear_regression", "random_forest", "svm"}

# Instance attributes

          def __init__(self, model_type, custom_params):

                    self.model_type: str = model_type
                    self.parameters = parameters
```

class PredictiveModels:

```
# Class attributes

        SUPPORTED_MODELS  = {"linear_regression", "random_forest", "svm"}

# Instance attributes

        def __init__(self, model_type, custom_params):

                self.model_type: str = model_type
                self.parameters = parameters

# Initialize instance methods: Behavior

        def _generate_model(self, parameters):
                if self.model_type == "linear_regression":
                        return LinearRegression(self.parameters)
                elif self.model_type == "random_forest":
                        return RandomForestRegressor(self.parameters)
                elif self.model_type == "svm":
                        return SVR(self.parameters)

        def fit_model(self, X, y): self.model.fit(X, y)
                return self.model

        def predict(self, X_new):
                if hasattr(self.model, 'predict'):
                        return self.model.predict(X_new)
                else: raise AttributeError("Model has not been trained or does not support 'predict' method.")
```

```
 # Instantiate new object (instance) from Class

model = PredictiveModels(model_type="random_forest", custom_params={"n_estimators": 50, "max_depth": 5}):


 # Access a instance attribute

  model.model_type


 # Access a Class attribute

  PredictiveModels.Supported_Models  # access via class name (convention)

  model.Supported_Models  # access via instance

# Invoke an instance method

  prediction = model.predict(X_test)
  print(prediction)
```