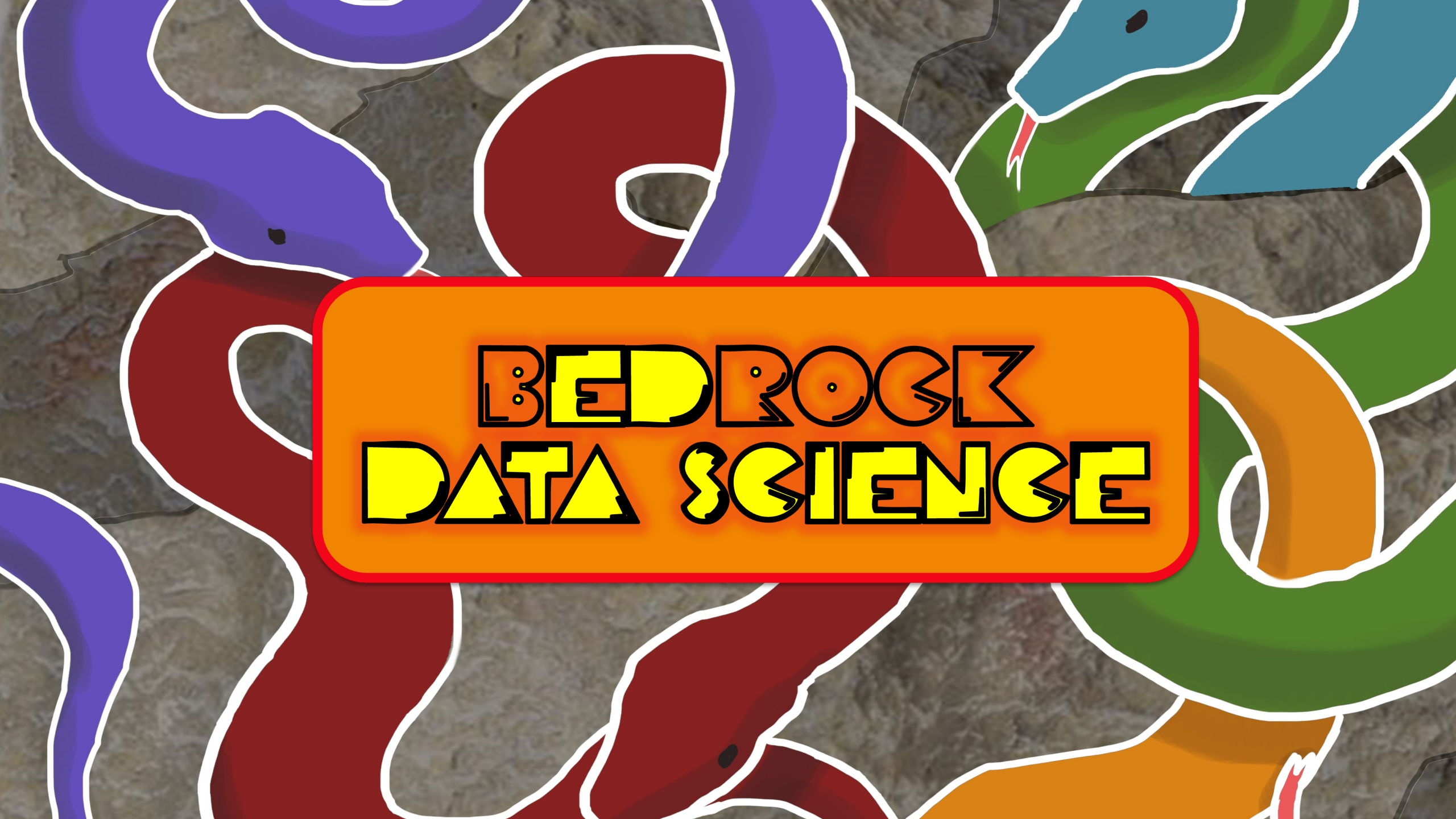

BEDROCK
DATA SCIENCE

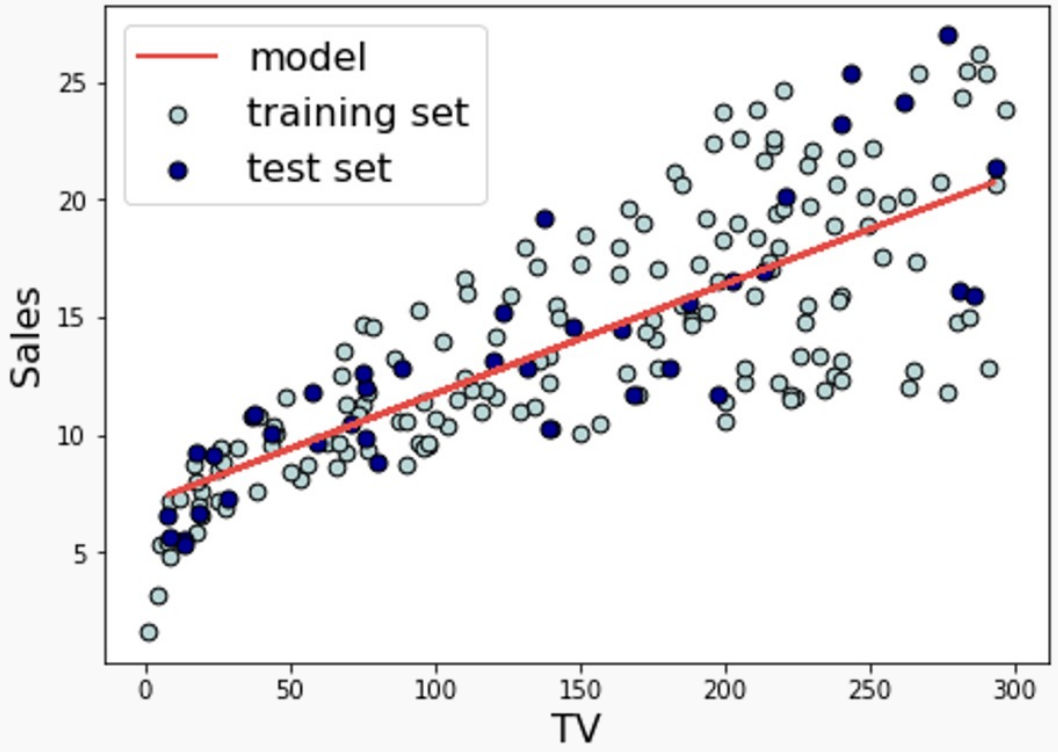
If you don't enjoy, you won't excel!


Our Goal

# CODE

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

# OUTPUT



*What's going on here?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

# What are *libraries ?*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is a dictionary ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'

x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is an array ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)

data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')


number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is len(x) ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']


# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is a list comprehension ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is a Boolean mask ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*How to write a python function ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask], y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What is a class constructor ?*

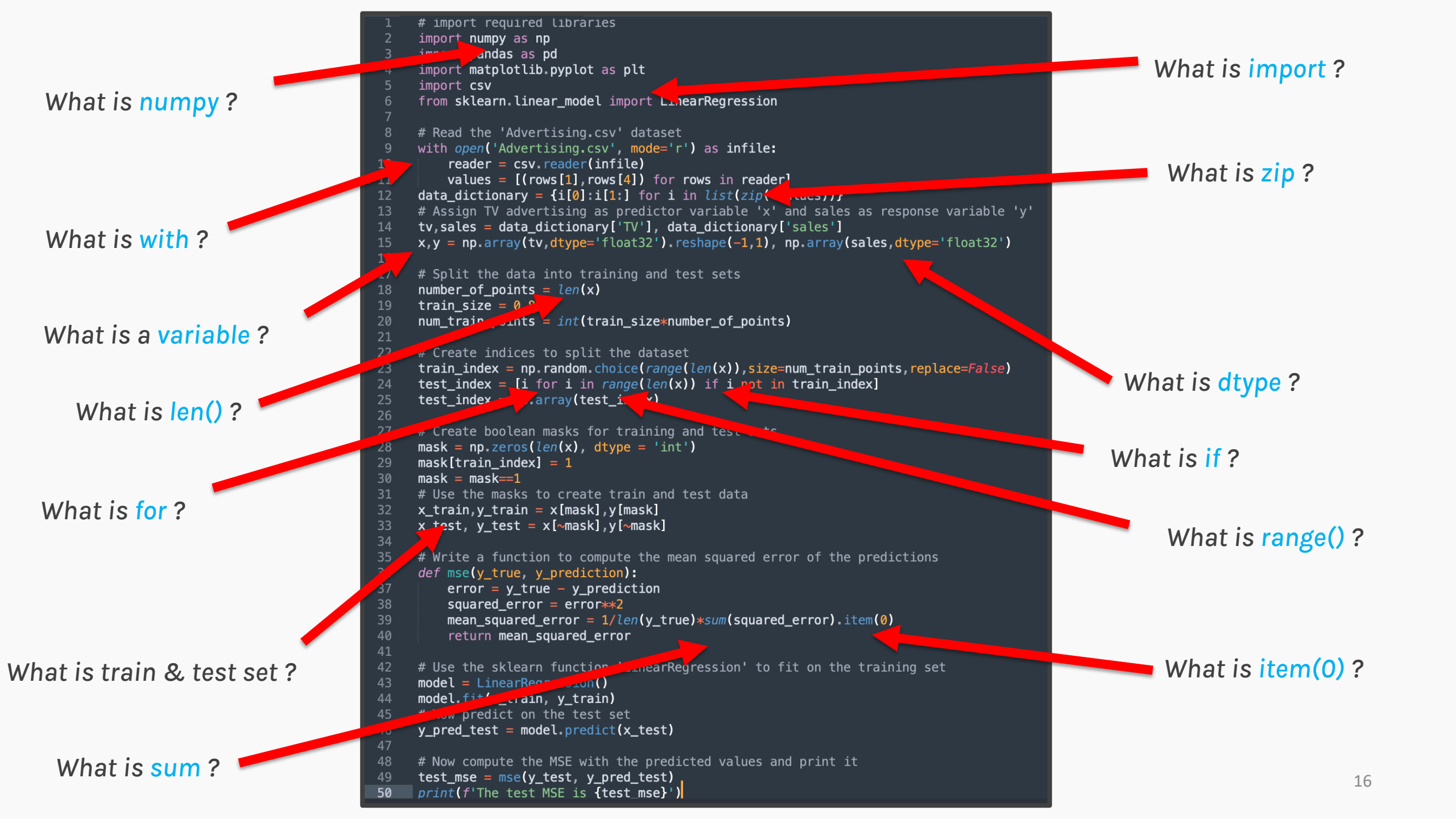
```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

*What does .fit() do ?*

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

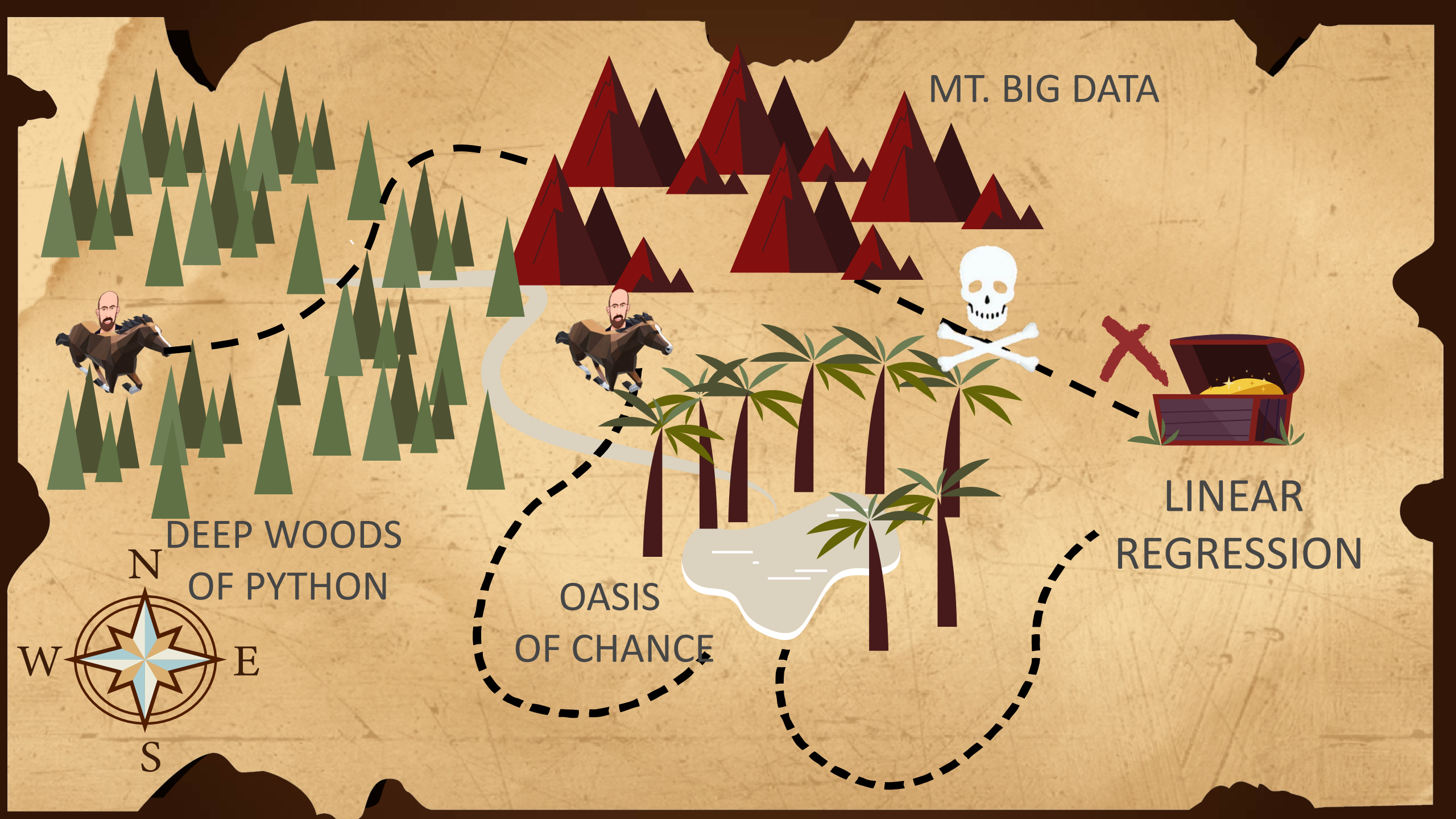
What does .predict() do ?

```python
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```



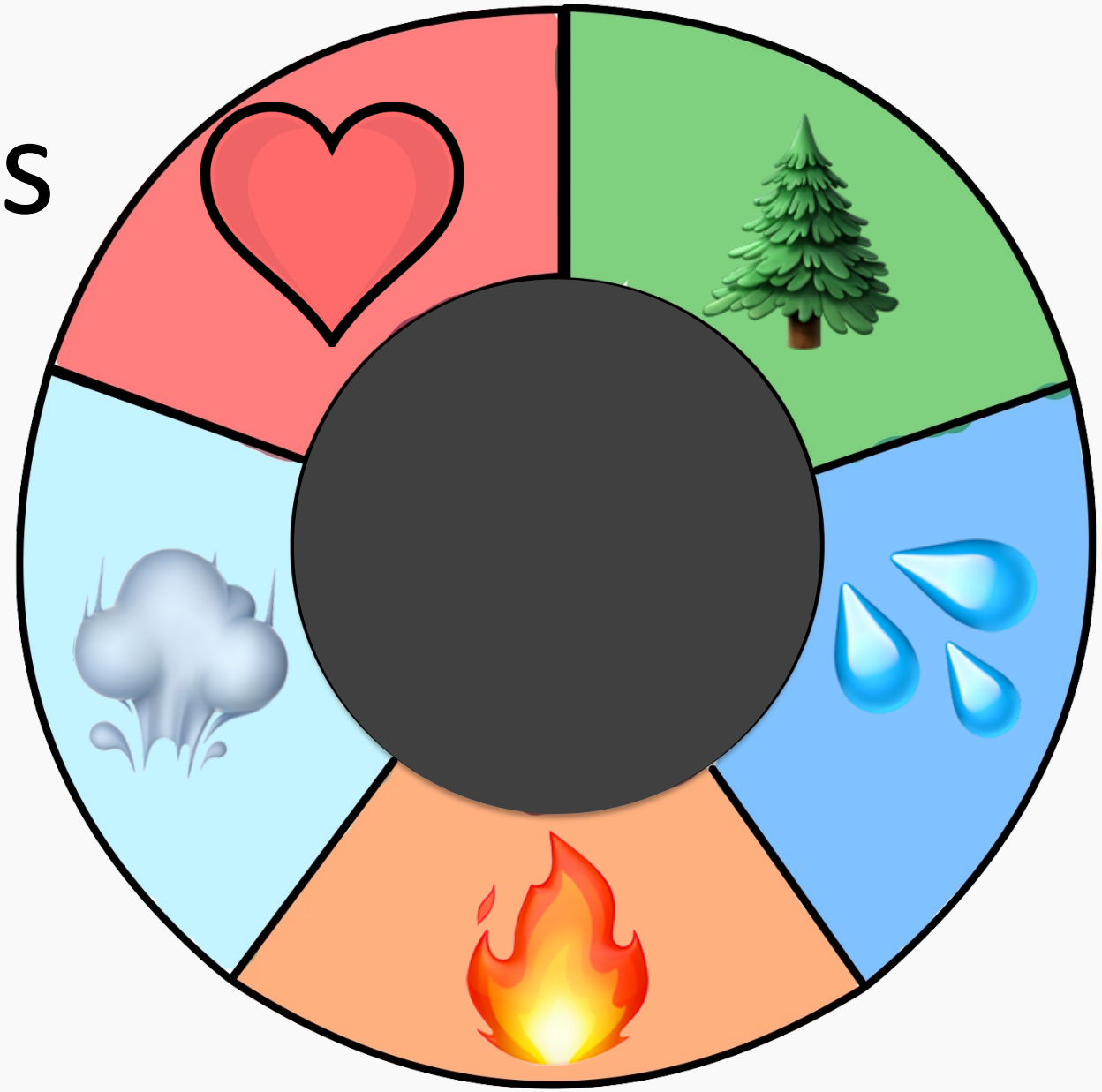

```
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv
from sklearn.linear_model import LinearRegression

# Read the 'Advertising.csv' dataset
with open('Advertising.csv', mode='r') as infile:
    reader = csv.reader(infile)
    values = [(rows[1],rows[4]) for rows in reader]
data_dictionary = {i[0]:i[1:] for i in list(zip(*values))}
# Assign TV advertising as predictor variable 'x' and sales as response variable 'y'
tv,sales = data_dictionary['TV'], data_dictionary['sales']
x,y = np.array(tv,dtype='float32').reshape(-1,1), np.array(sales,dtype='float32')

# Split the data into training and test sets
number_of_points = len(x)
train_size = 0.8
num_train_points = int(train_size*number_of_points)

# Create indices to split the dataset
train_index = np.random.choice(range(len(x)),size=num_train_points,replace=False)
test_index = [i for i in range(len(x)) if i not in train_index]
test_index = np.array(test_index)

# Create boolean masks for training and test sets
mask = np.zeros(len(x), dtype = 'int')
mask[train_index] = 1
mask = mask==1
# Use the masks to create train and test data
x_train,y_train = x[mask],y[mask]
x_test, y_test = x[~mask],y[~mask]

# Write a function to compute the mean squared error of the predictions
def mse(y_true, y_prediction):
    error = y_true - y_prediction
    squared_error = error**2
    mean_squared_error = 1/len(y_true)*sum(squared_error).item(0)
    return mean_squared_error

# Use the sklearn function 'LinearRegression' to fit on the training set
model = LinearRegression()
model.fit(x_train, y_train)
# Now predict on the test set
y_pred_test = model.predict(x_test)

# Now compute the MSE with the predicted values and print it
test_mse = mse(y_test, y_pred_test)
print(f'The test MSE is {test_mse}')
```

| SESSION | TOPICS COVERED |
| --- | --- |
| 1 | Introduction to Python Programming |
| 2 | Python Data Structures |
| 3 | Python Functions |
| 4 | Python Classes |
| 5 | Files, Strings and Regular Expressions |
| 6 | Numerical Python (NumPy) |
| 7 | Linear Algebra |
| 8 | Introduction to Probability |
| 9 | Panel Data (Pandas) |
| 10 | Calculus & Linear Regression |


MT. BIG DATA
DEEP WOODS
OF PYTHON
OASIS
OF CHANCE
LINEAR
REGRESSION
N
W
E
S

Strings

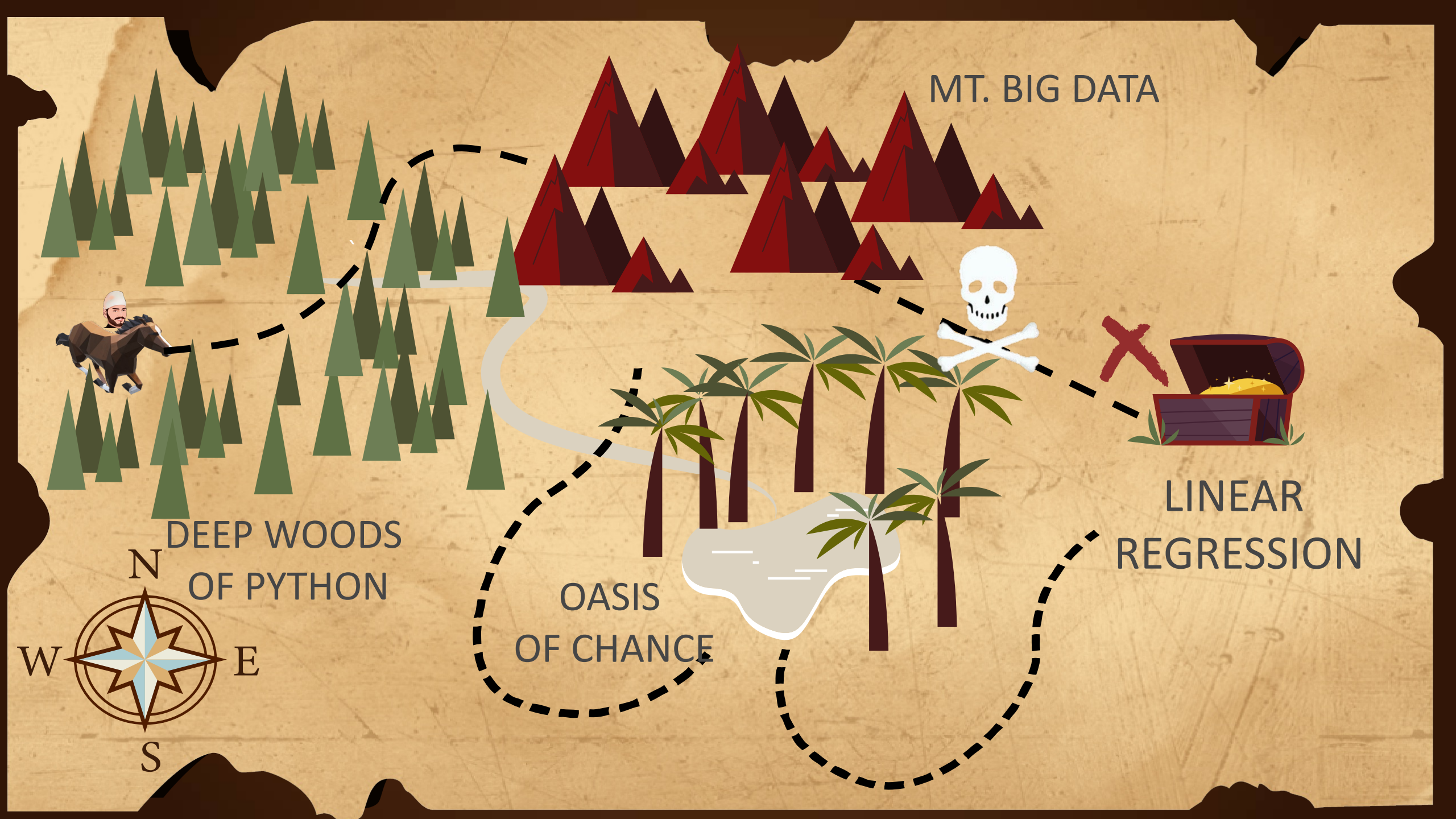
Lists



classes

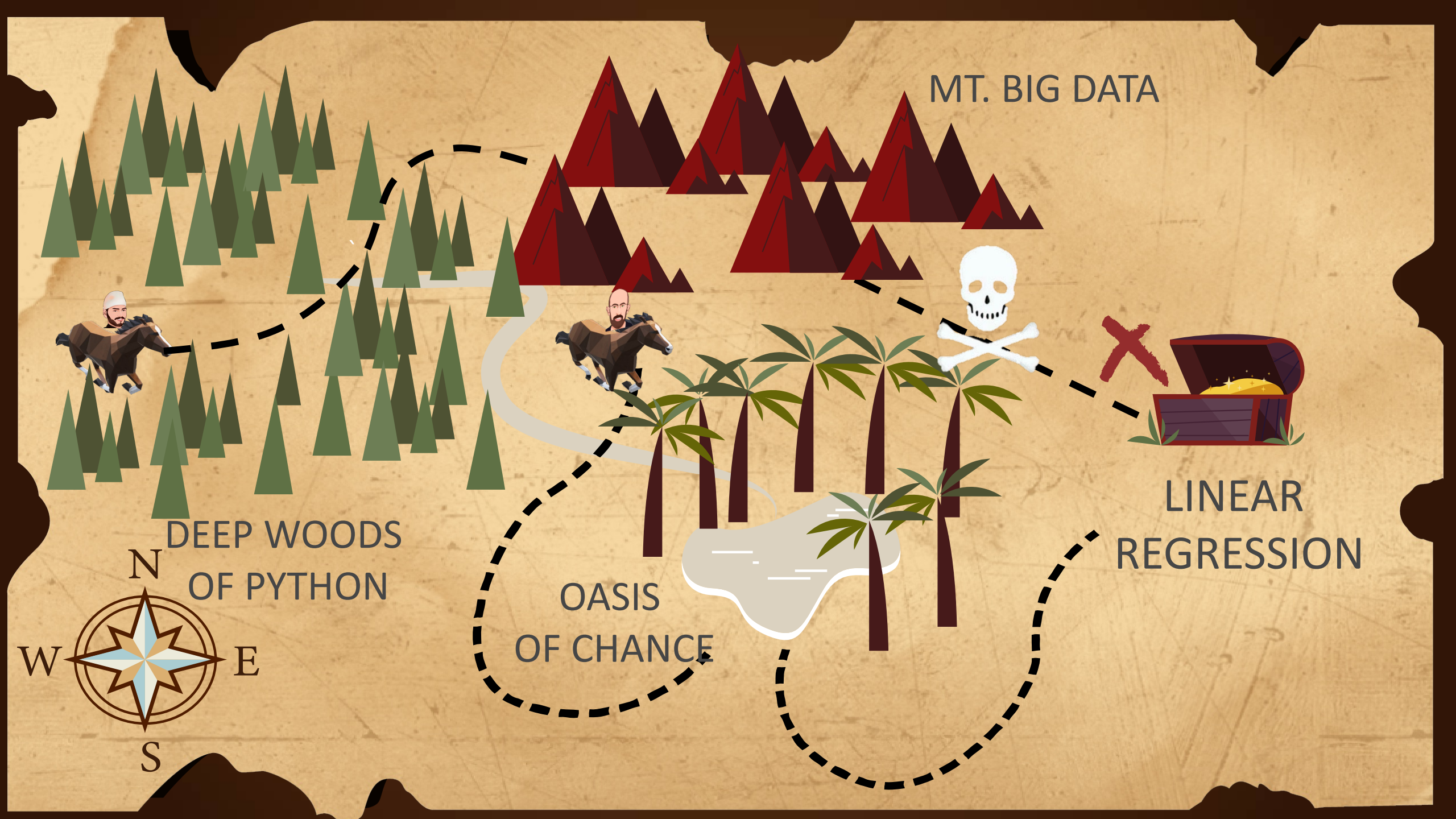
dictionary

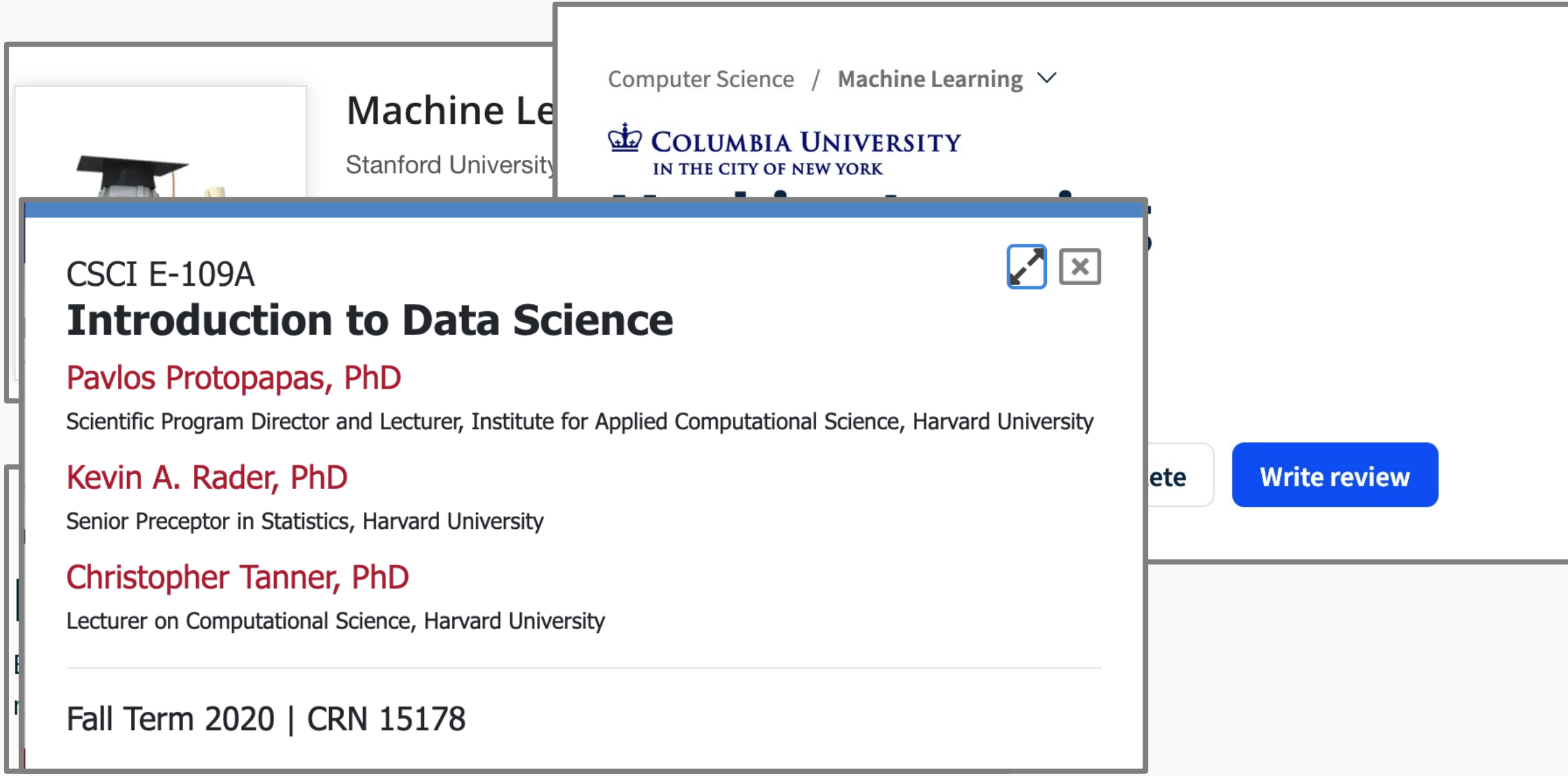
Functions


MT. BIG DATA
DEEP WOODS
OF PYTHON
OASIS
OF CHANCE
LINEAR
REGRESSION
N
W
E
S

pandas

NumPy


MT. BIG DATA
DEEP WOODS
OF PYTHON
OASIS
OF CHANCE
LINEAR
REGRESSION
N
W
E
S


Machine Le
Stanford University
Computer Science / Machine Learning
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK
ete
Write review
CSCI E-109A
Introduction to Data Science
Pavlos Protopapas, PhD
Scientific Program Director and Lecturer, Institute for Applied Computational Science, Harvard University
Kevin A. Rader, PhD
Senior Preceptor in Statistics, Harvard University
Christopher Tanner, PhD
Lecturer on Computational Science, Harvard University
Fall Term 2020 | CRN 15178

# *Our Schedule*

# *Our Expectation*

👉 *Complete sessions*

👉 *Complete session quiz and exercises*

👉 *Ask and answer questions on the forum*


“Your Journey Begins Here!”

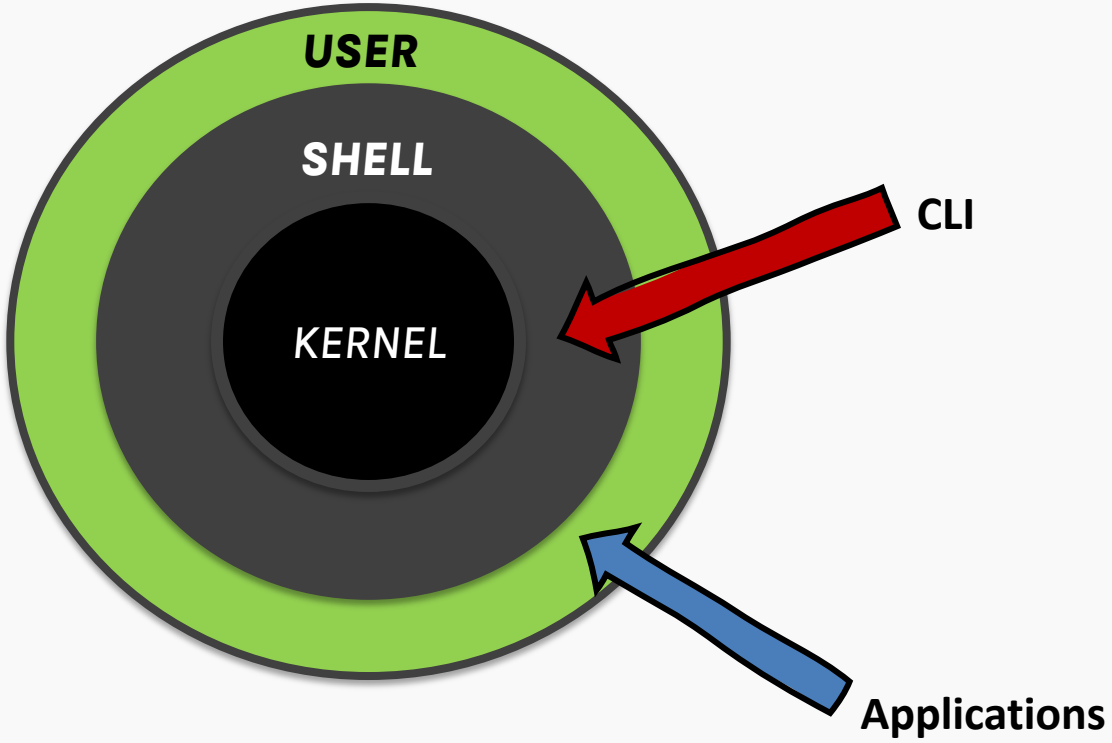
# Let's begin

~~*What is Python?*~~

# What is the *Terminal*?

# What is the terminal/Unix Shell ?

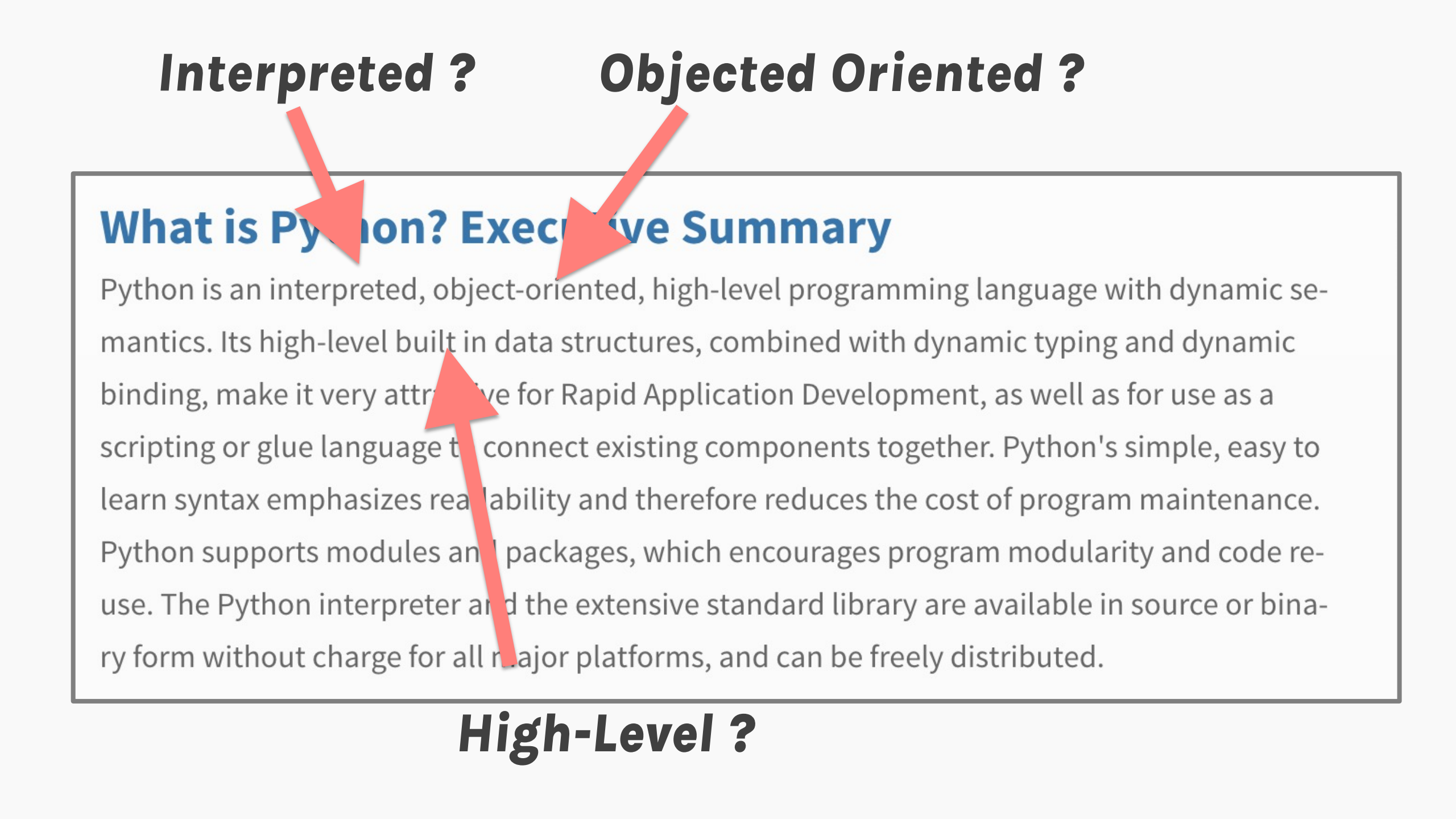
- *A command line interface to a machine*
- *Controls the execution of the system*




# *What is Python?*

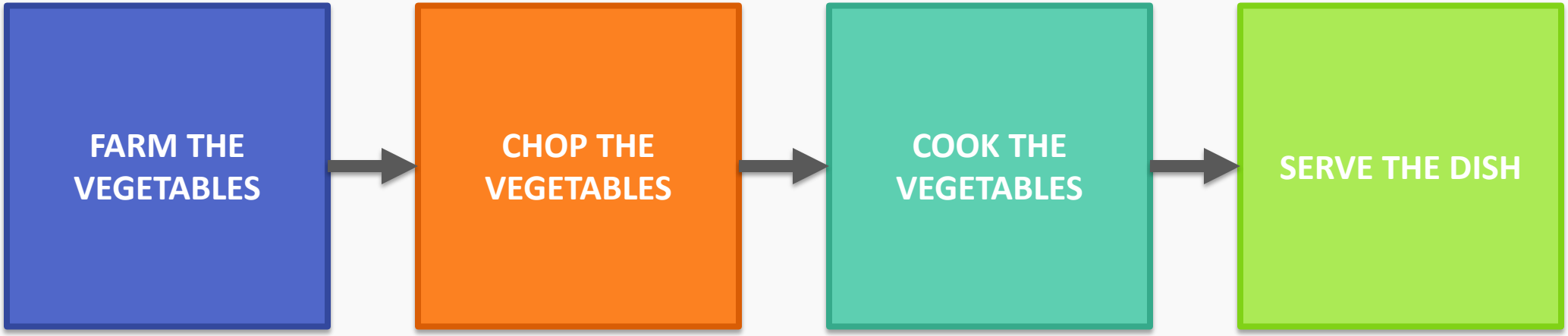

## What is Python? Executive Summary

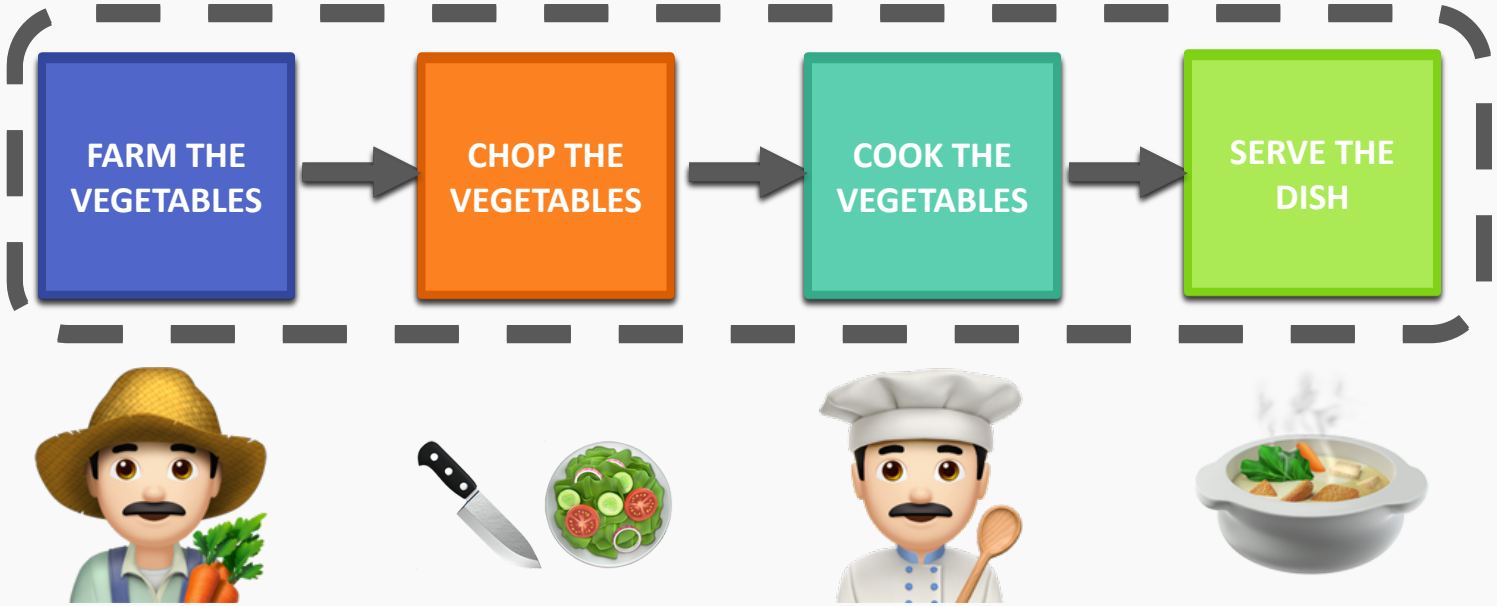
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.


Interpreted ?
Objected Oriented ?
What is Python? Executive Summary
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
High-Level ?

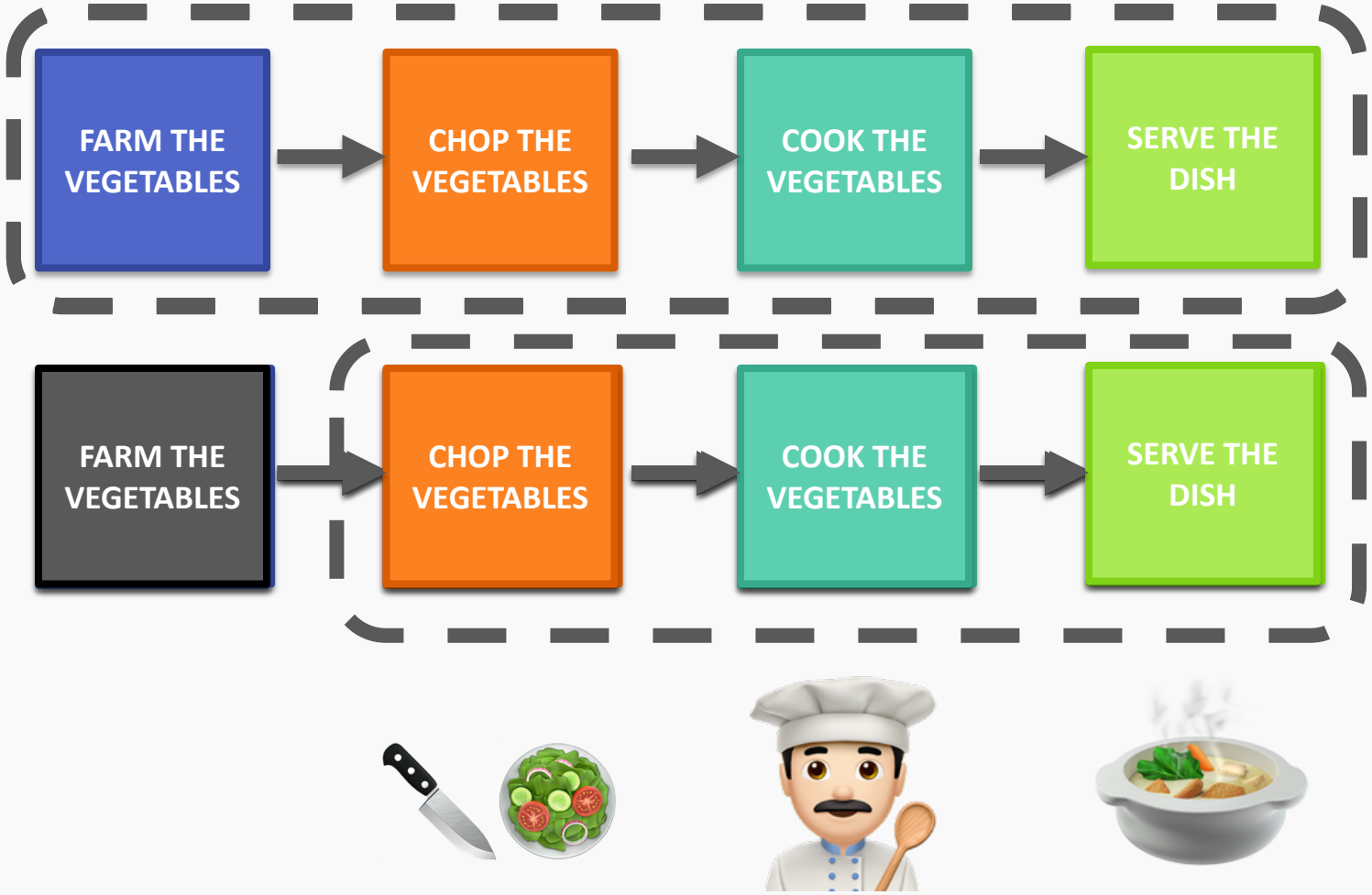
*Let’s break it down*

*Think of programming as preparing a meal*


FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH

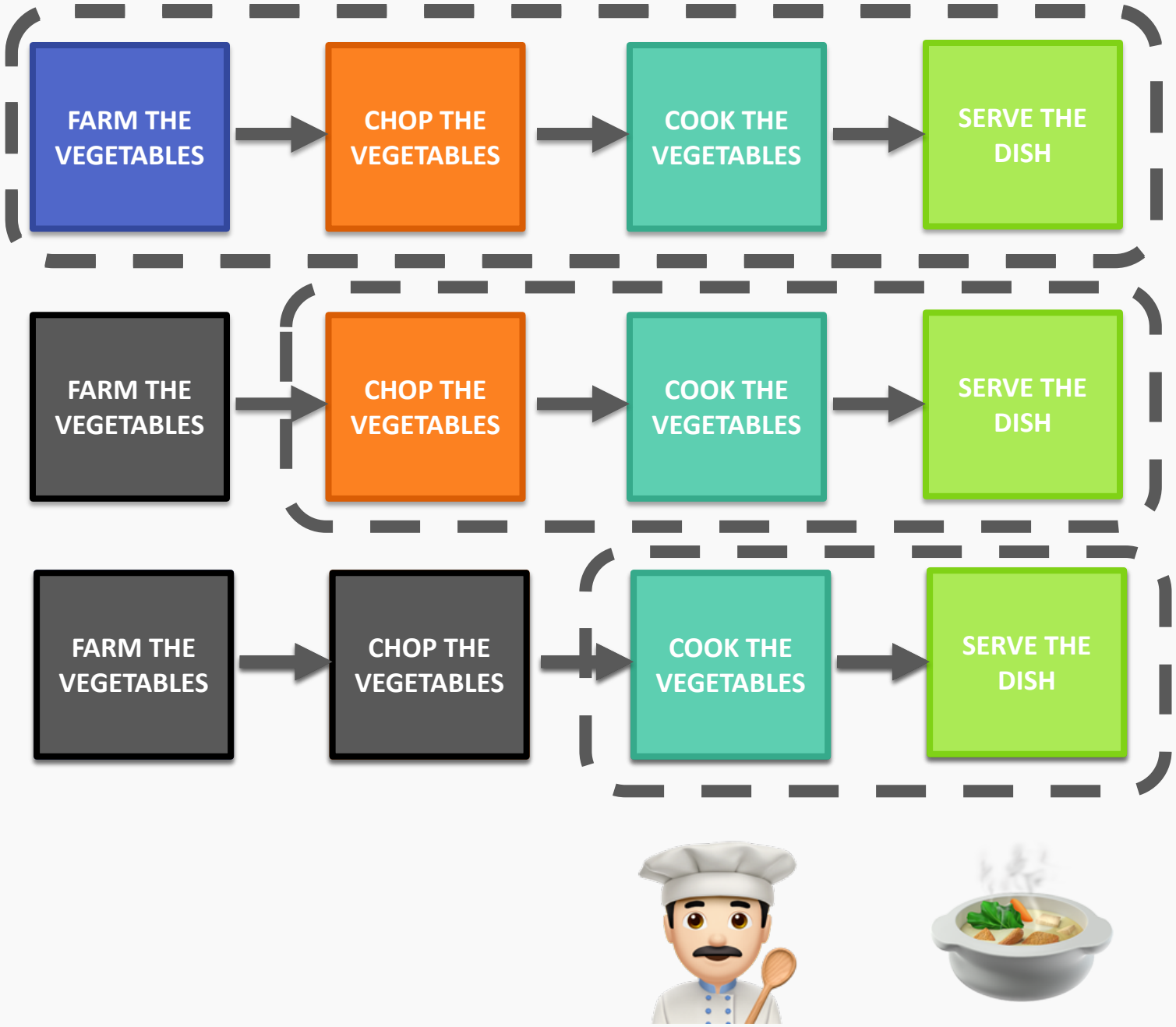



*Assembly Level Code*




*Assembly Level Code*
*(Machine Level)*

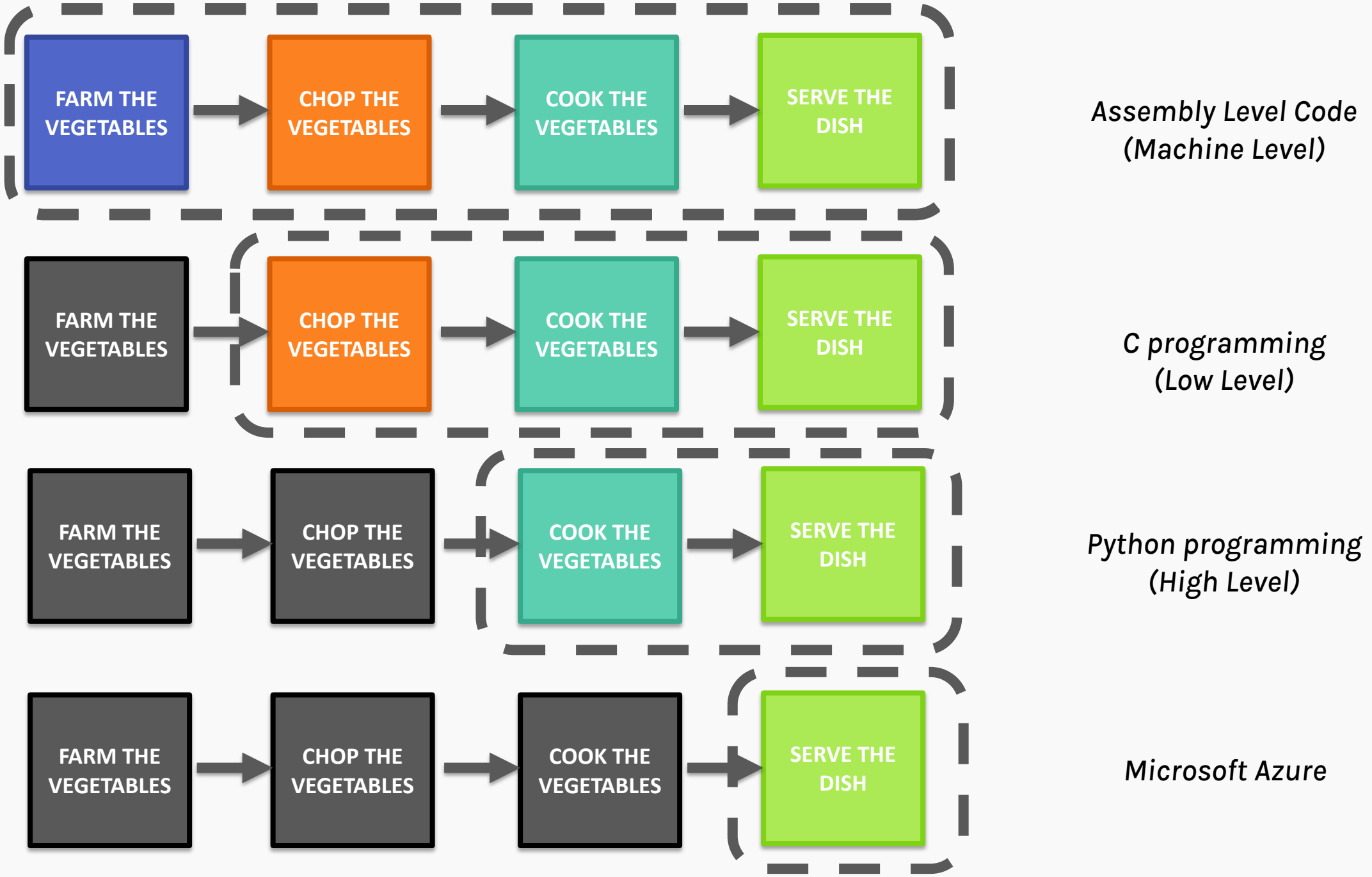
*C programming*
*(Low Level)*



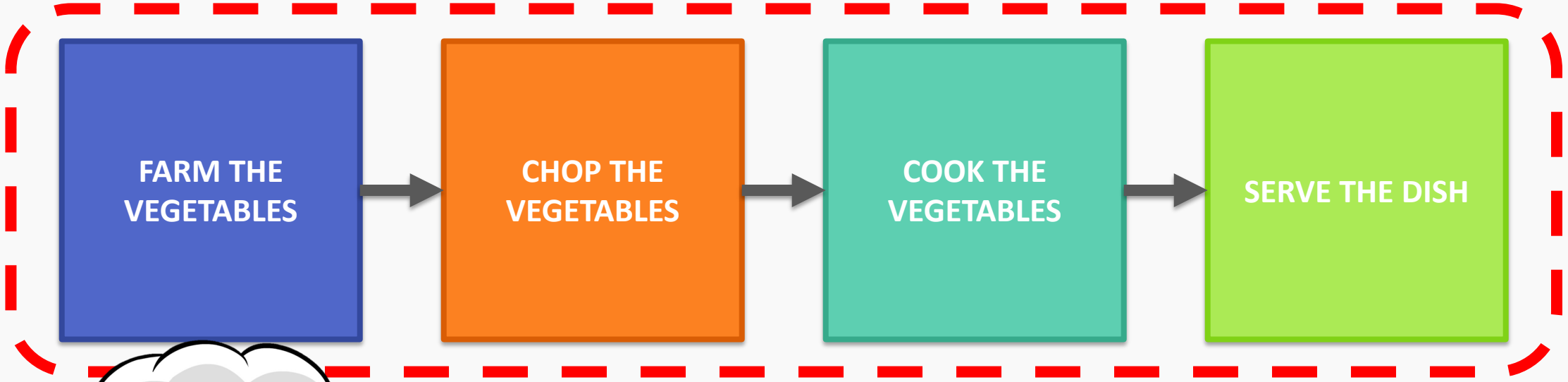

*Assembly Level Code*
*(Machine Level)*

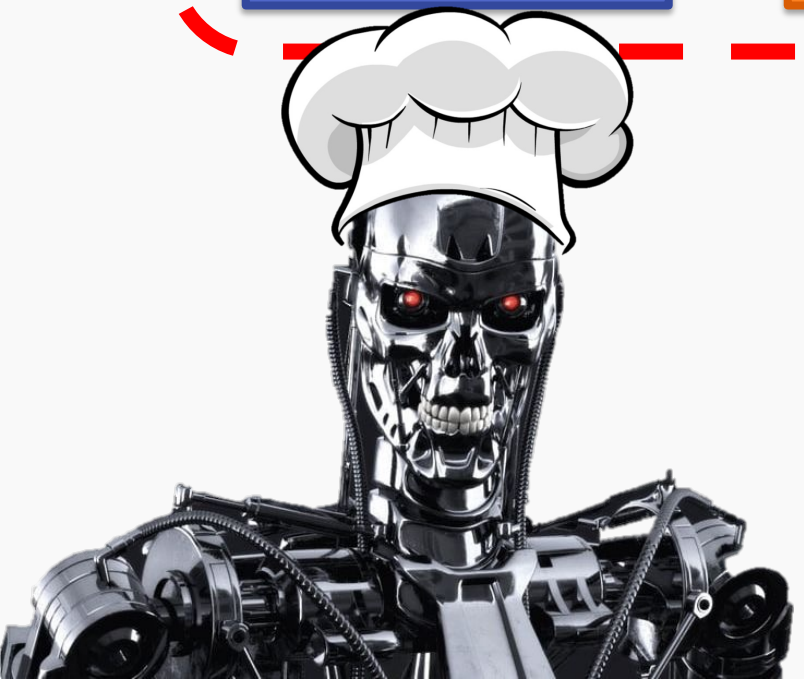
*C programming*
*(Low Level)*

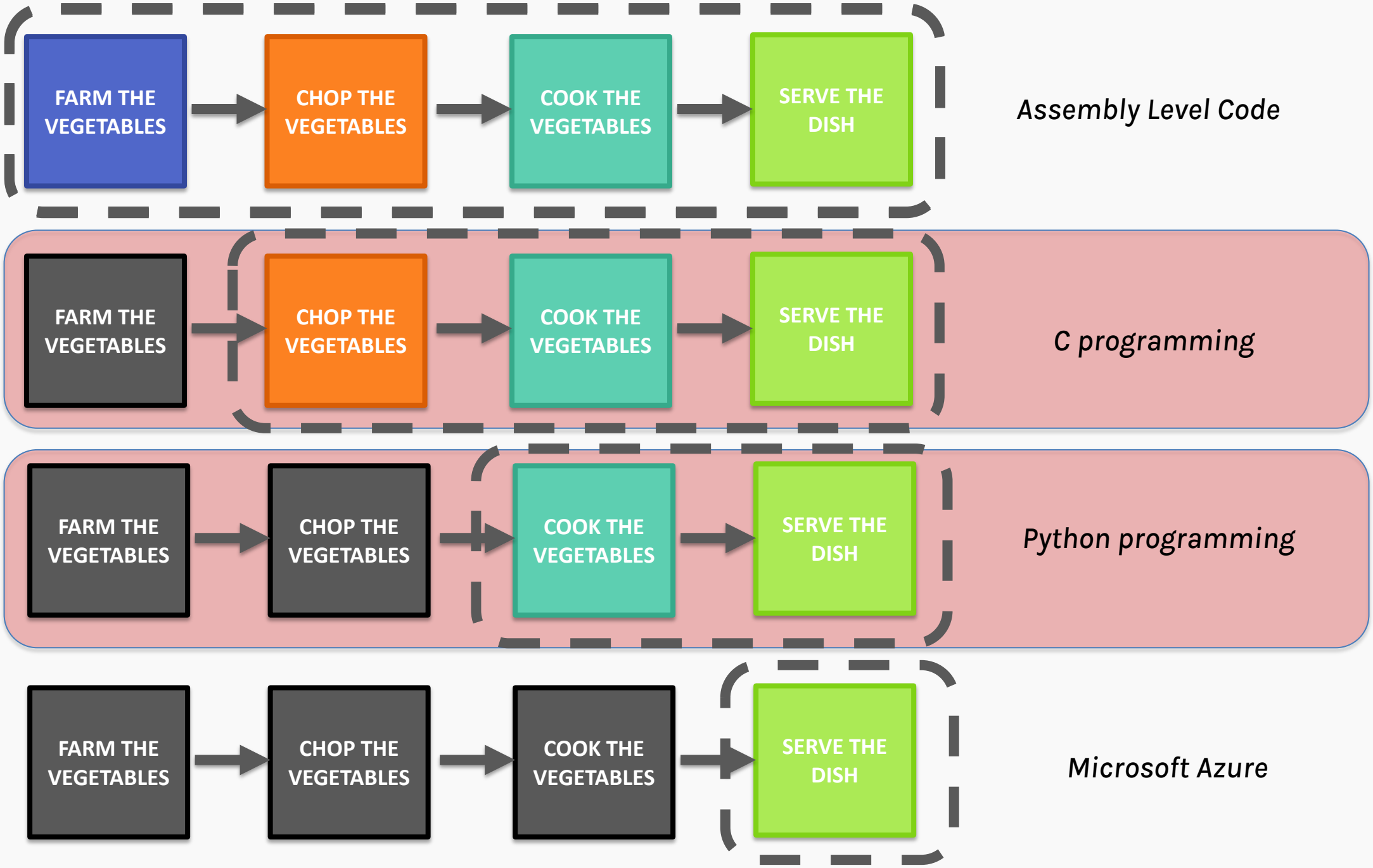
*Python programming*
*(High Level)*


FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Assembly Level Code (Machine Level)
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
C programming (Low Level)
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Python programming (High Level)
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Microsoft Azure

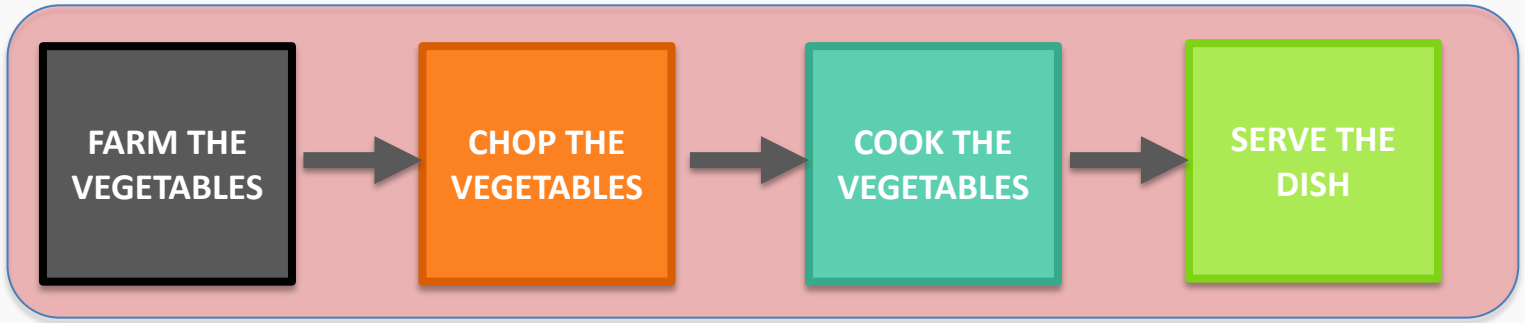
# AI Apocalypse



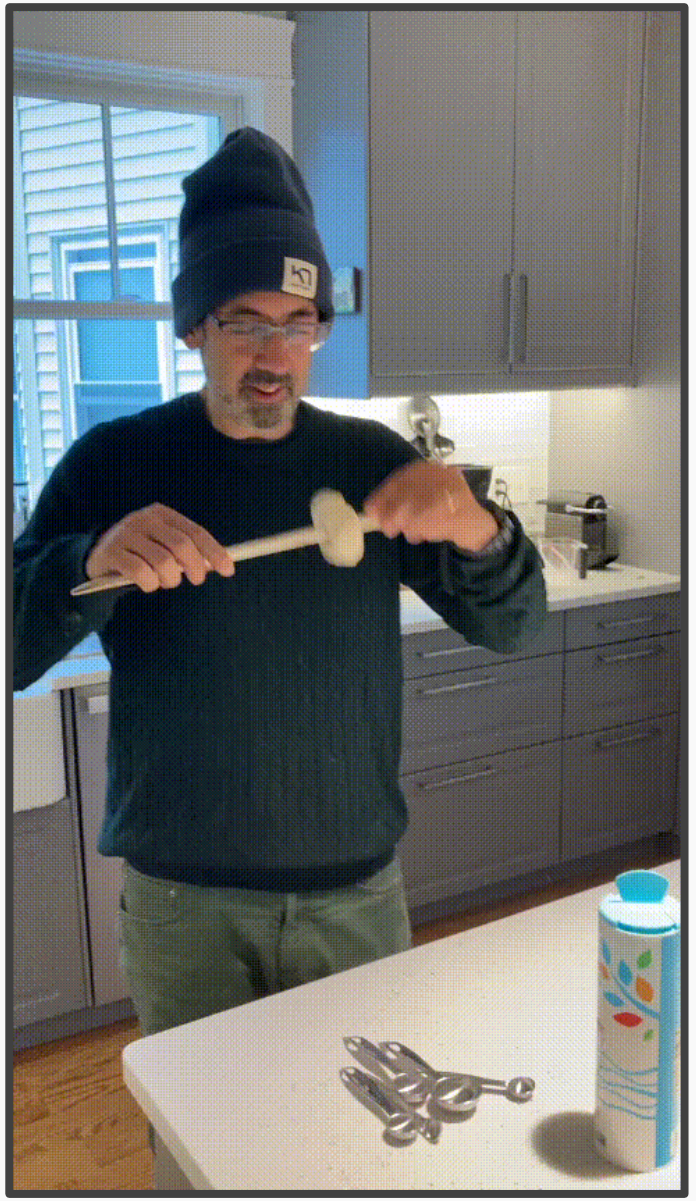




FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Assembly Level Code
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
C programming
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Python programming
FARM THE VEGETABLES
CHOP THE VEGETABLES
COOK THE VEGETABLES
SERVE THE DISH
Microsoft Azure

# Cook your own meal




- *Making a meal from scratch is not easy, but it gives you a lot of control.*
- *The C-programming language is as such. It is a compiled language (can be directly converted to machine code for execution).*
- *C is fast but requires low-level design choices (type casting, memory allocation)*



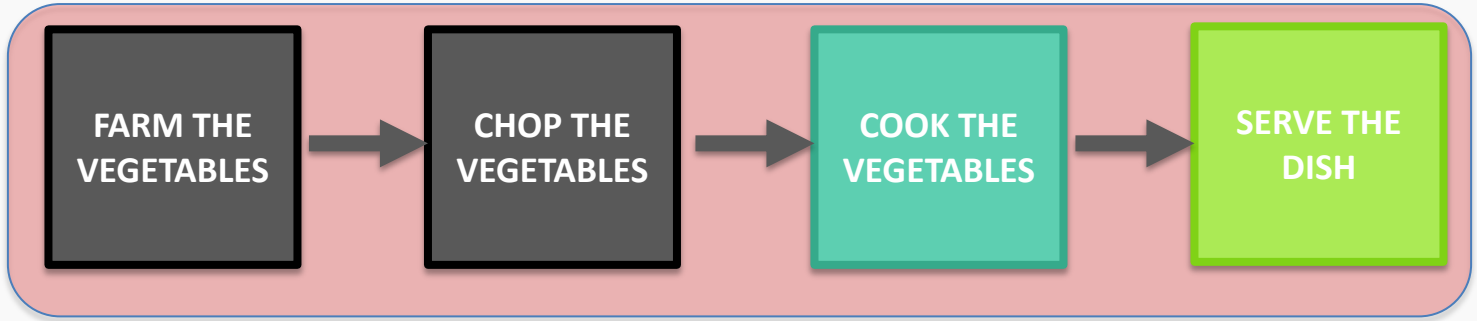
# C programming

## C ISSUES?

- *C is still the language of choice for low-level applications such as embedded systems design.*
- *However, it is painstakingly verbose for data science applications.*
- *On the right is an example code to print a list of integers, needing several lines of code.*

```c
#include <stdio.h>
void main()
{
int arr[10];
int i;
printf("\n\nRead and Print elements of an
array:\n");
printf("-----------------------------------
-\n");
printf("Input 10 elements in the array :\n");
for(i=0; i<10; i++)
{
printf("element - %d : ",i);
scanf("%d", &arr[i]);
}
printf("\nElements in array are: ");
for(i=0; i<10; i++)
{
printf("%d ", arr[i]);
}
printf("\n");
```

# Meal-kits




- *Imagine getting all ingredients pre-chopped and ready to be used.*
- *The Python programming language is as such. It is an interpreted language.*
- *This saves the user from taking all the technical design decisions (memory management, type-casting etc.)*

# Python source code vs C source code

```c
#include <stdio.h>
void main()
{
int arr[10];
int i;
printf("\n\nRead and Print elements of an
array:\n");
printf("-------------------------------------
-\n");
printf("Input 10 elements in the array :\n");
for(i=0; i<10; i++)
{
printf("element - %d : ",i);
scanf("%d", &arr[i]);
}
printf("\nElements in array are: ");
for(i=0; i<10; i++)
{
printf("%d ", arr[i]);
}
printf("\n");
```

```python
print("Input 10 elements in the
array")
num_elements = [int(input(f"Element
{i+1}: ")) for i in range(10)]
print(f"The elements in the array
are: {num_elements}")
```

# Python source code

**Python ISSUES?**

```
from pytube import YouTube
Youtube(url).streams.first().download(location)
```

- *Python takes away the arduous steps of machine micro-management.*
- *However, all instructions must be provided before execution. (**ISSUE #1**)*
- *As Python handles the low-level decisions for you, there is a cost overhead for execution, making it slow.* **(ISSUE #2)***

*You just need two lines of code to download youtube videos using python*

# IPython

- *Any good chef knows the importance of tasting the meals while preparing a meal.*
- *The **Interactive Python** (IPython) project is as such. It is a READ-EVAL-PRINT-LOOP environment*
- *This is especially useful for experimentation.*
- *First introduced in 2001, IPython has become extremely popular among data scientists.*

# IPython

## IPython ISSUES?

- *IPython runs on the command line, it becomes easy to lose track of operations.*
- *It is also difficult to organize your code and share it with others.*
- *To overcome such issues, a web-app, called **Project Jupyter** was implemented to make it user-friendly & collaborative.*

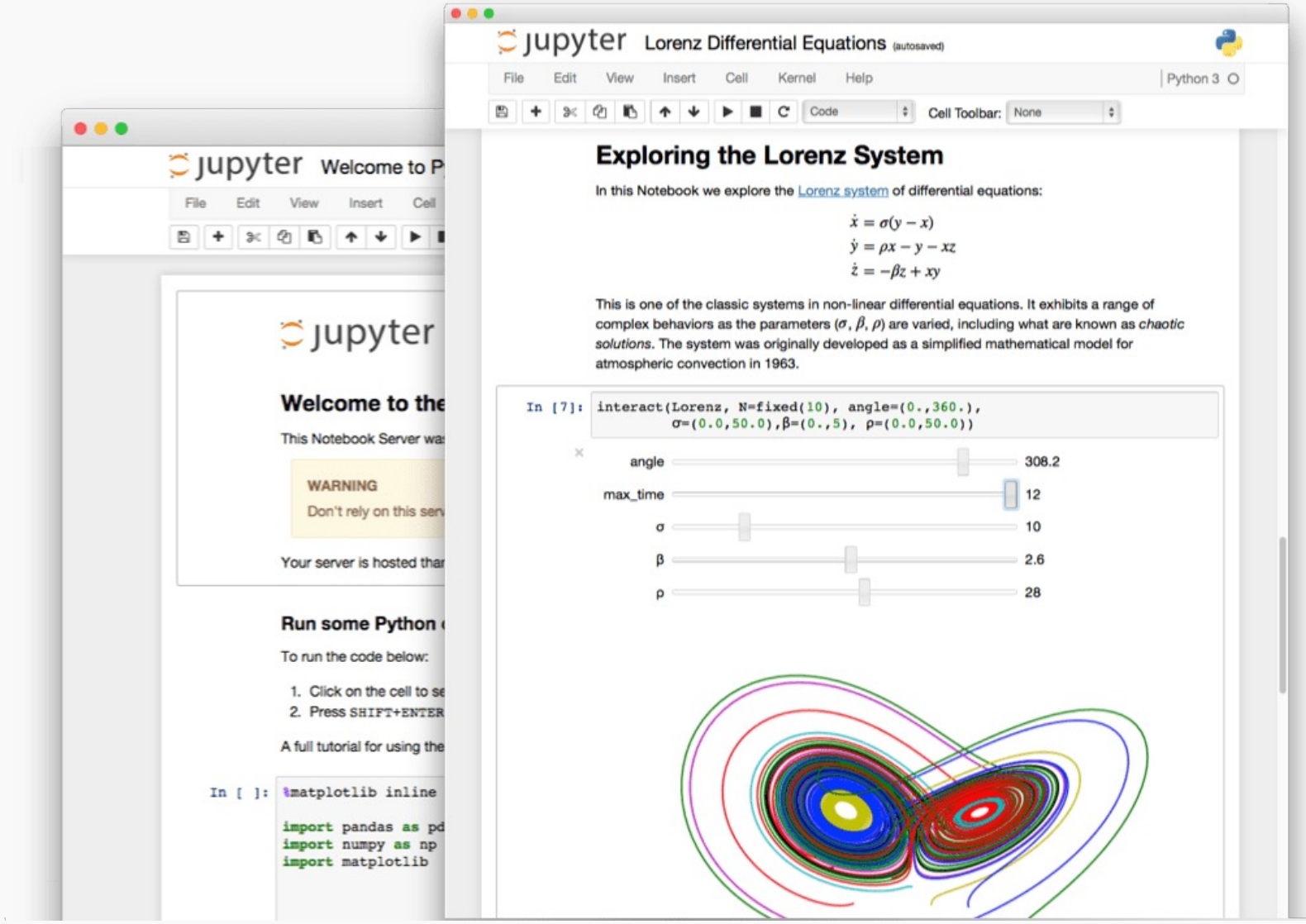
```
[]$ ipython
Python 3.8.3 (default, Jul  2 2020, 11:26:31)
Type 'copyright', 'credits' or 'license' for mor
e information
IPython 7.16.1 -- An enhanced Interactive Python
. Type '?' for help.

In [1]: num_elements = 5

In [2]: new_array = [i for i in range(num_e
   ...: lements)]

In [3]: print(f'The elements in the array a
   ...: re {new_array}')
The elements in the array are [0, 1, 2, 3, 4]

In [4]:
```

*IPython will run on your terminal,unix shell,command prompt*

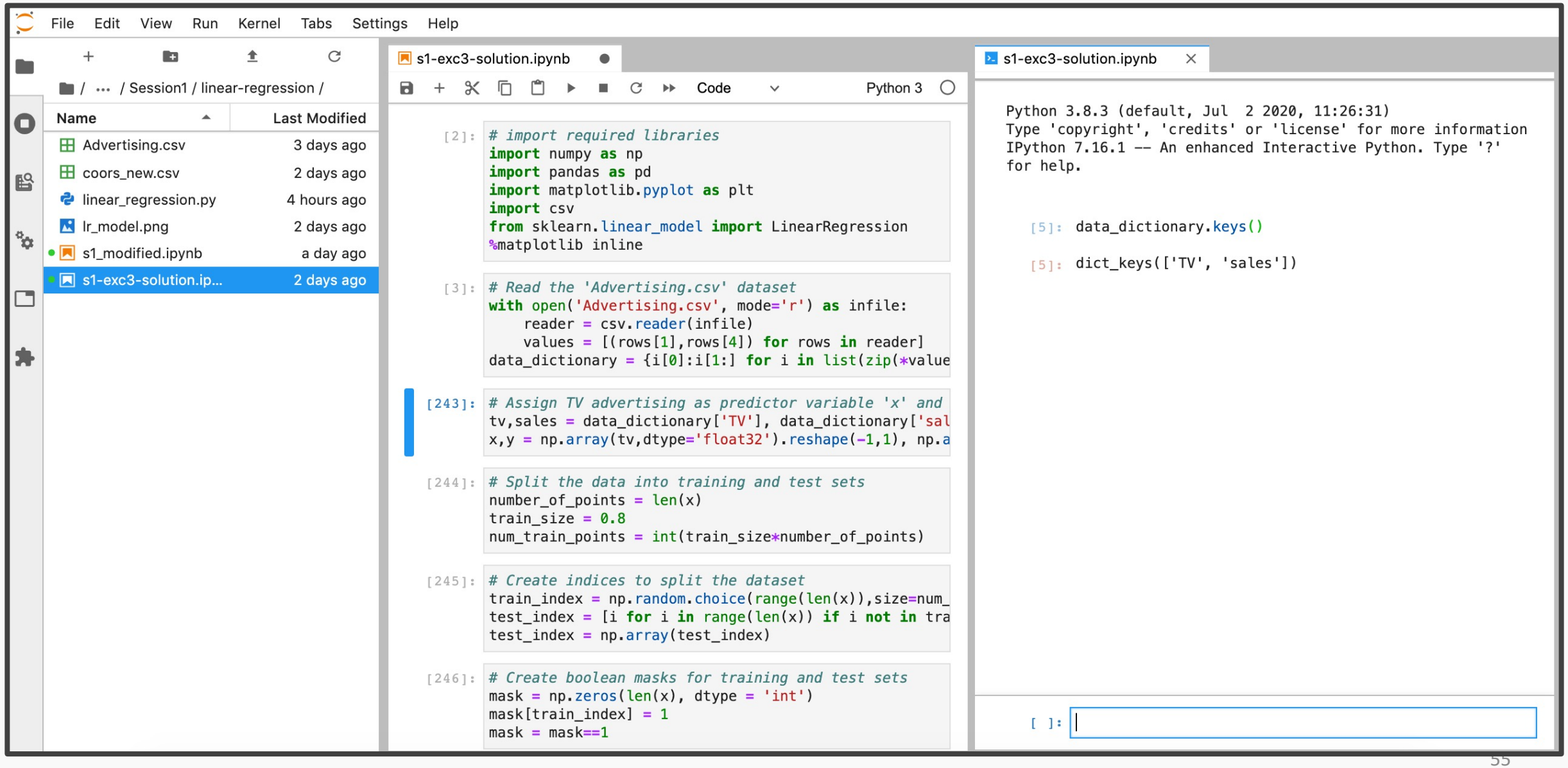
*Project Jupyter*

# Experimental Programming with Jupyter

- *Project Jupyter offers a web-app interface where you can create notebooks that can your code cell-by-cell.*
- *This feature allows you to easily experiment with your code and make corrections as you go.*
- *Jupyter notebooks can be easily shared with others just like pdf files.*

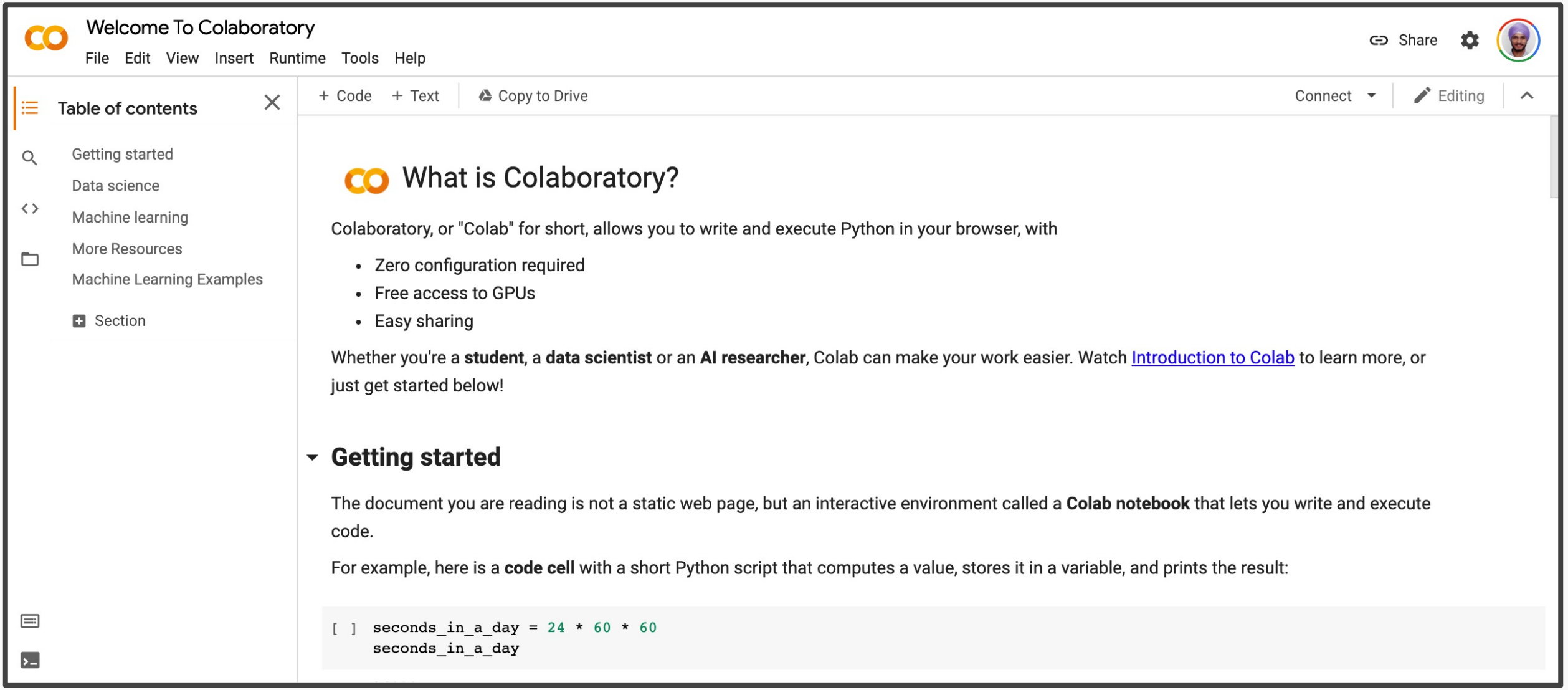


# Jupyter Lab interface

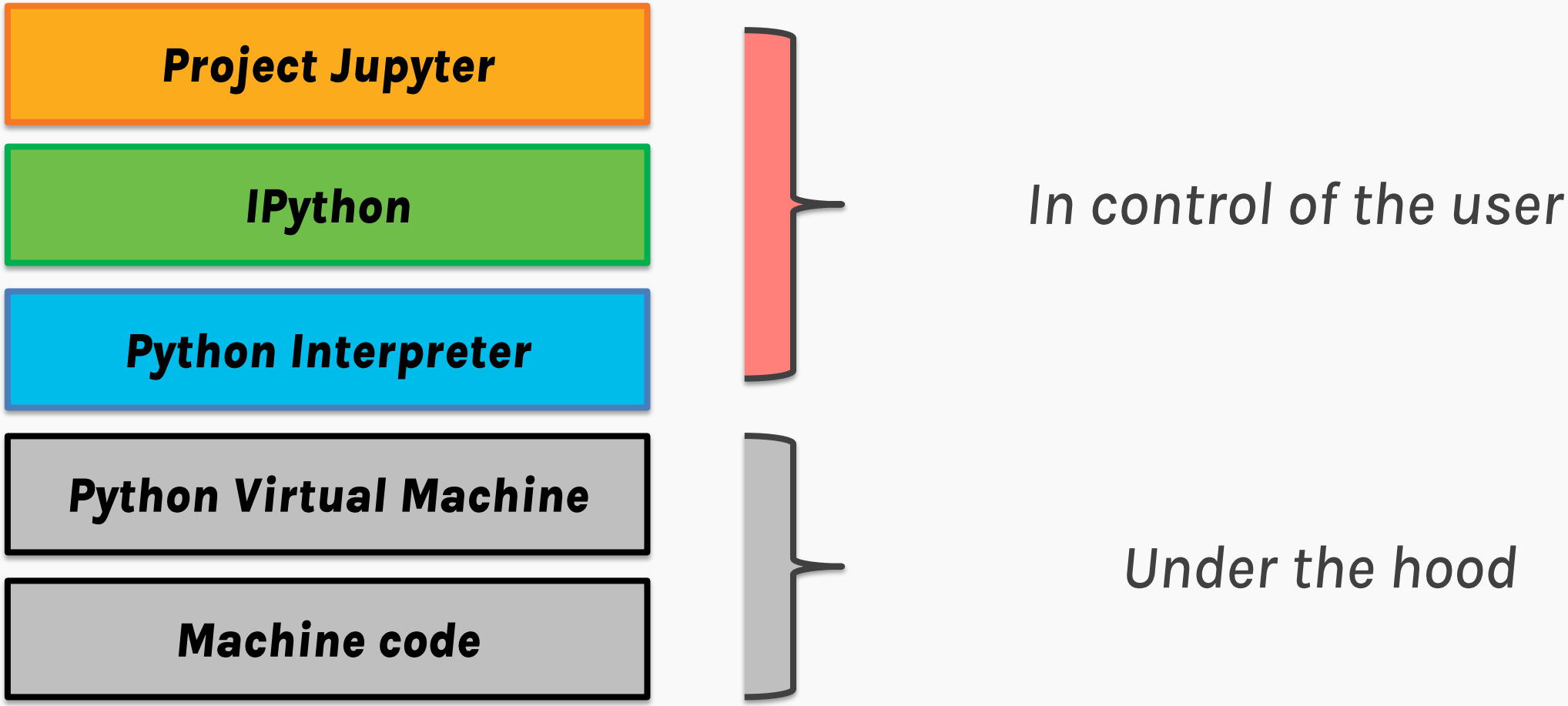


# Google Colaboratory

## *Google's custom implementation of Project Jupyter*



# Bringing it all together



# My Specials

*The key concepts of programming, math and statistics that you learn in BRDS will be relevant throughout your journey as a Data Scientist*

jupyter

**BEDROCK**

*Code first introduction*

**CS109A CS109B**

| *Machine learning* | *Computer Vision* | *Natural Language Processing* | *Generative Models* |
|---|---|---|---|

**AC215**

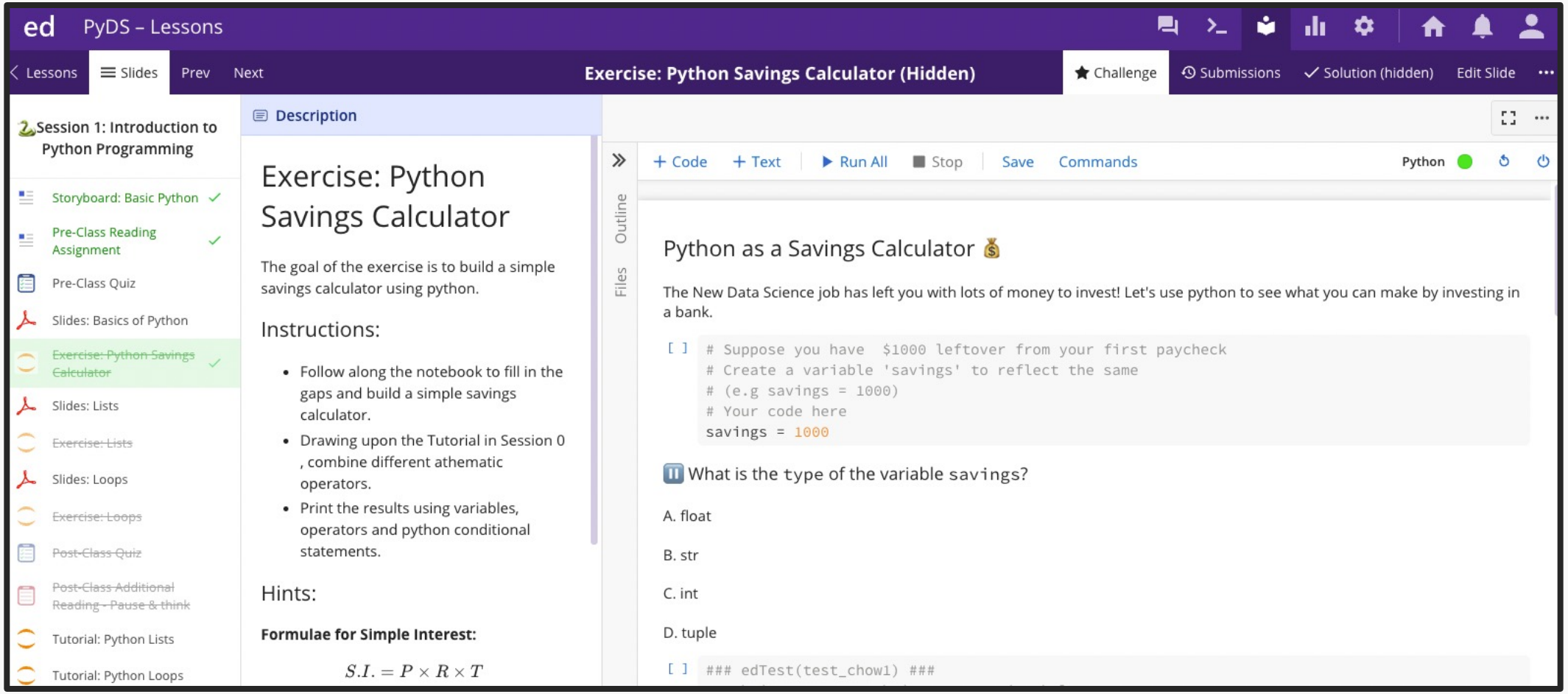
*Application Development & Deployment*

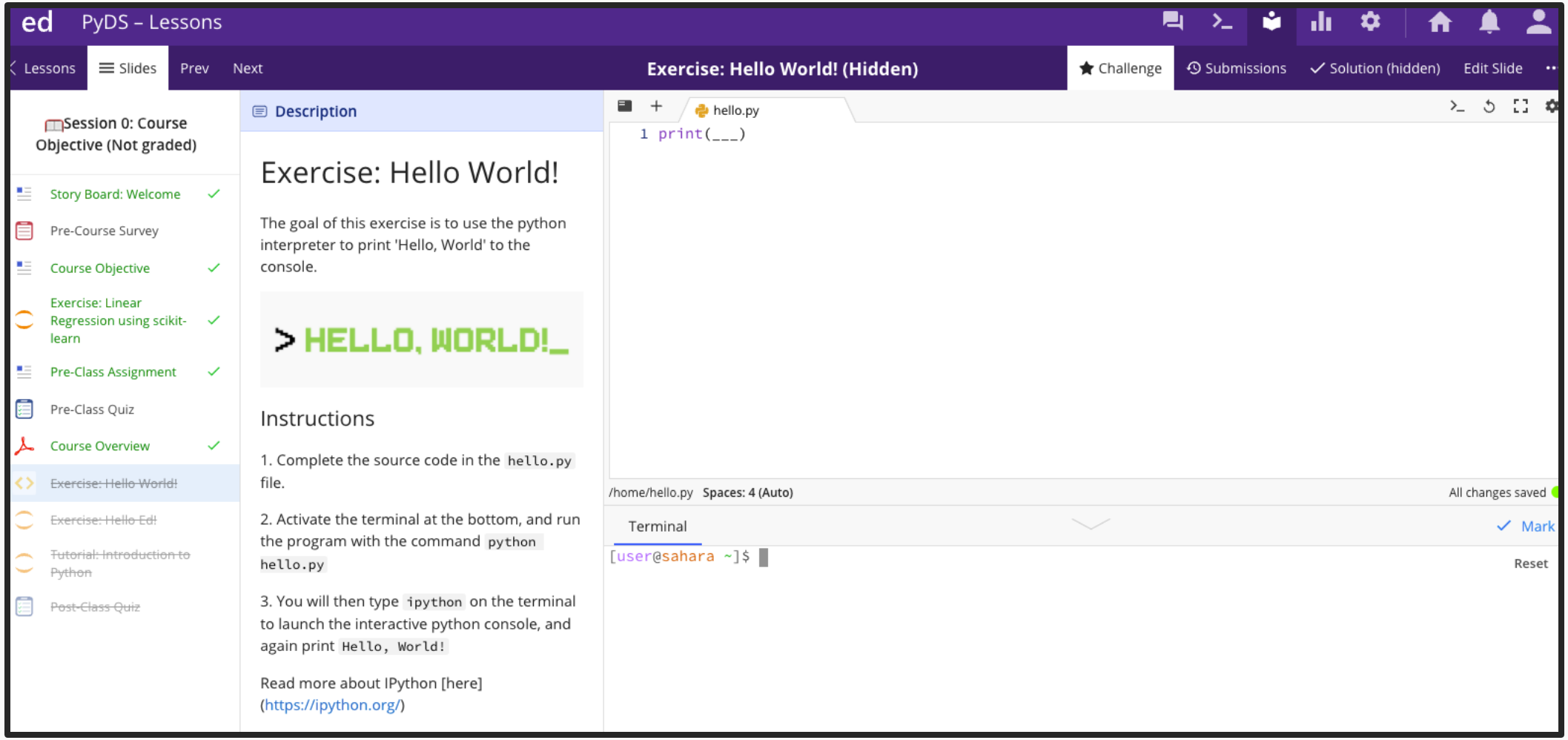
# *Programming on your own system*

- *There are several ways to get project Jupyter on your system, (e.g.* ***Anaconda)***
- *However, beginners generally hesitate to use local environments.*
- *We will be using Ed-Stem instead so that you can learn to program without the fear of breaking something.*

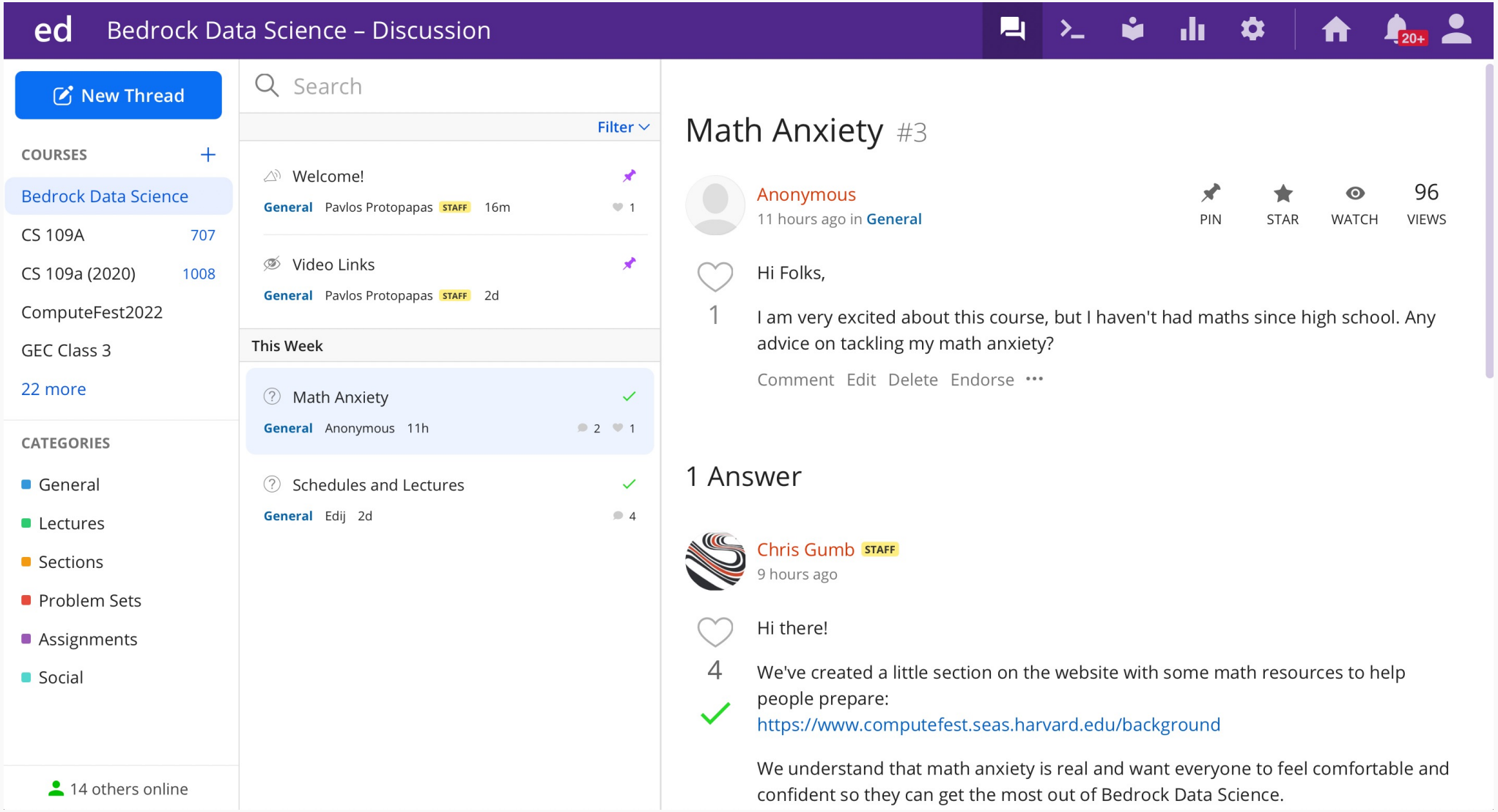
# *What is EdStem?*

# *EdStem: An integrated platform*



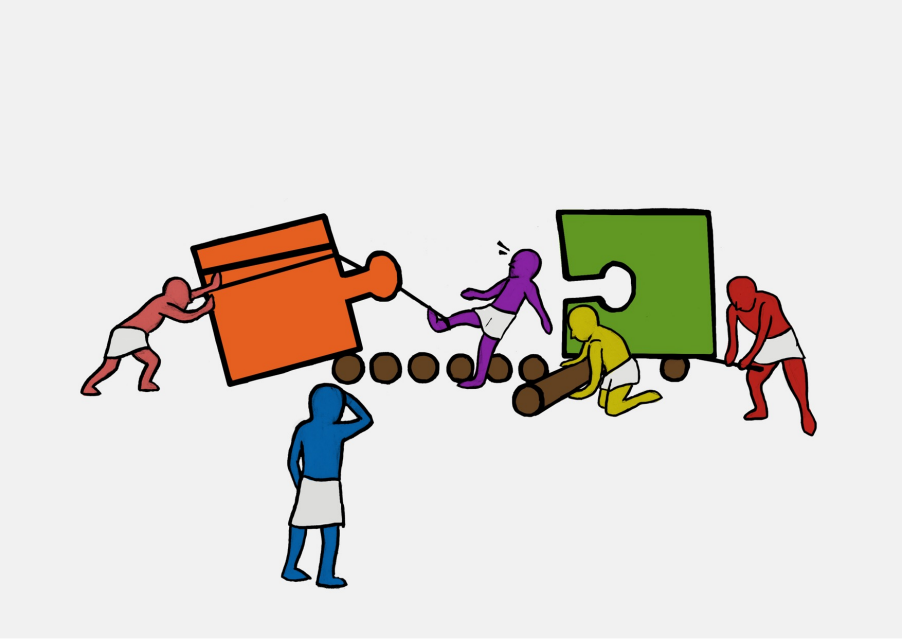

ed
PyDS – Lessons
Lessons
Slides
Prev
Next
Exercise: Hello World! (Hidden)
Challenge
Submissions
Solution (hidden)
Edit Slide
Session 0: Course Objective (Not graded)
Story Board: Welcome
Pre-Course Survey
Course Objective
Exercise: Linear Regression using scikit-learn
Pre-Class Assignment
Pre-Class Quiz
Course Overview
Exercise: Hello World!
Exercise: Hello Ed!
Tutorial: Introduction to Python
Post-Class Quiz
Description
Exercise: Hello World!
The goal of this exercise is to use the python interpreter to print 'Hello, World' to the console.
> HELLO, WORLD!_
Instructions
1. Complete the source code in the hello.py file.
2. Activate the terminal at the bottom, and run the program with the command python hello.py
3. You will then type ipython on the terminal to launch the interactive python console, and again print Hello, World!
Read more about IPython [here] (https://ipython.org/)
hello.py
1 print(___)
/home/hello.py Spaces: 4 (Auto)
All changes saved
Terminal
Mark
[user@sahara ~]$
Reset

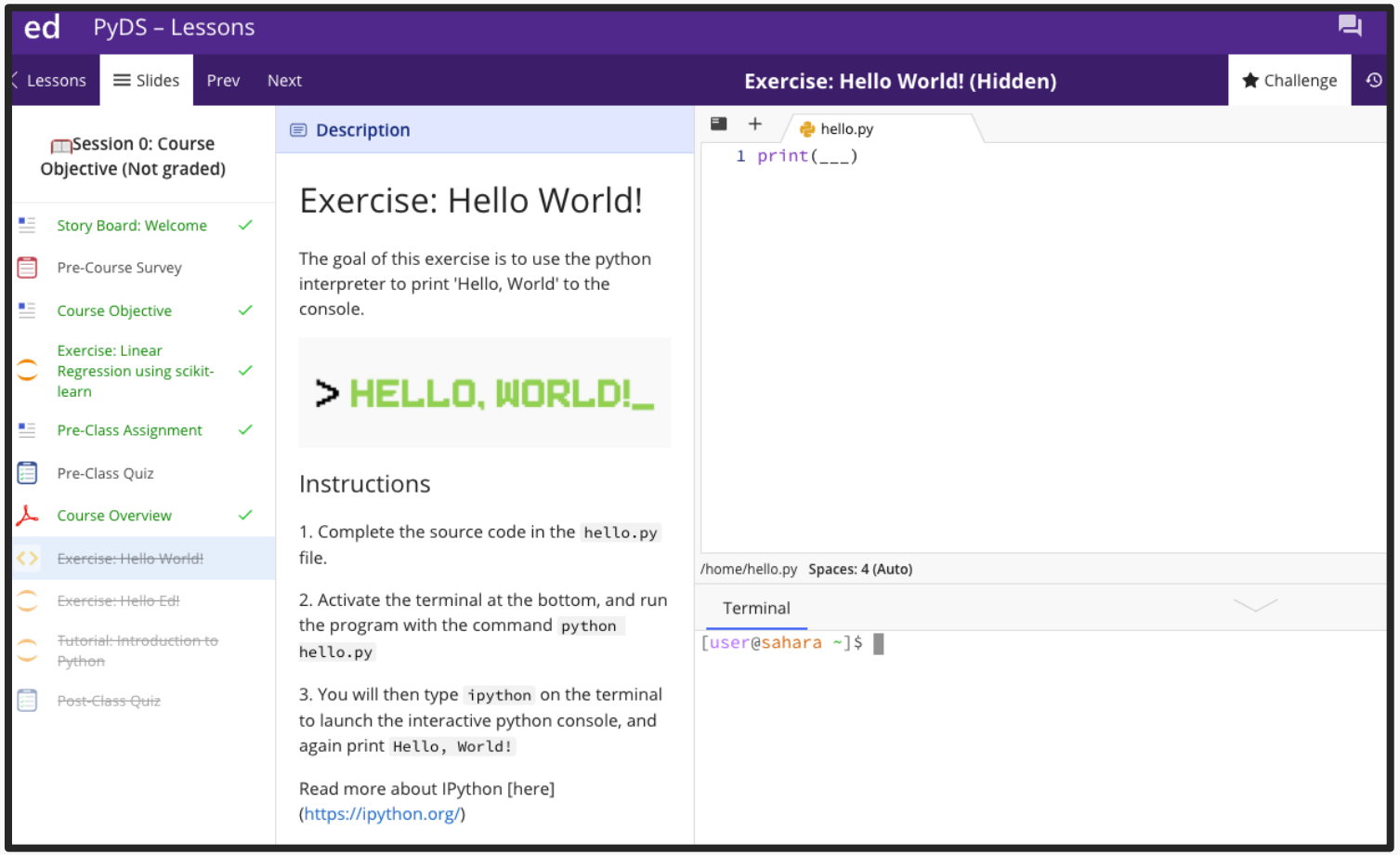
# Discussion forums for students



# Exercise

> Hello, World!_





# ANY QUESTIONS ?