In [47]:
```python
# Importing Libraries
import sqlite3

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
```

In [48]:
```python
#Loafing the database into dataframe
c = sqlite3.connect('database.sqlite')

# List all tables in the database
df = pd.read_sql_query("SELECT * FROM sqlite_master WHERE type='table'",c)

# Output dataframe
df
```

Out[48]:

	type	name	tbl_name	rootpage	sql
0	table	sqlite_sequence	sqlite_sequence	4	CREATE TABLE sqlite_sequence(name,seq)
1	table	Player_Attributes	Player_Attributes	11	CREATE TABLE "Player_Attributes" (\n\t\tIN...
2	table	Player	Player	14	CREATE TABLE $Player$ (\n\t\tINTEGER PRIMA...
3	table	Match	Match	18	CREATE TABLE $Match$ (\n\t\tINTEGER PRIMAR...
4	table	League	League	24	CREATE TABLE $League$ (\n\t\tINTEGER PRIMA...
5	table	Country	Country	26	CREATE TABLE $Country$ (\n\t\tINTEGER PRIM...
6	table	Team	Team	29	CREATE TABLE "Team" (\n\t\tINTEGER PRIMARY...
7	table	Team_Attributes	Team_Attributes	2	CREATE TABLE $Team_Aributes$ (\n\t\tINTE...

In [49]:
```python
player_attr_df = pd.read_sql("SELECT * FROM Player_Attributes",c) # Input Code Here
player_attr_df.fillna(11, inplace=True)
#Display first 5 rows
```

In [50]:
```python
player_attr_df.head(5)
```

Out[50]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attacking_work_rate	defensive_work_rate
0	1	218353	505942	2016-02-18 00:00:00	67.0	71.0	right	medium	medium
1	2	218353	505942	2015-11-19 00:00:00	67.0	71.0	right	medium	medium
2	3	218353	505942	2015-09-21 00:00:00	62.0	66.0	right	medium	medium
3	4	218353	505942	2015-03-20 00:00:00	61.0	65.0	right	medium	medium
4	5	218353	505942	2007-02-22 00:00:00	61.0	65.0	right	medium	medium

5 rows × 42 columns

In [51]:
```python
len(player_attr_df)
```

Out[51]:
```
183978
```

In [52]:
```python
player_attr_df.columns
```

Out[52]:
```
Index(['id', 'player_fifa_api_id', 'player_api_id', 'date', 'overall_rating',
       'potential', 'preferred_foot', 'attacking_work_rate',
       'defensive_work_rate', 'crossing', 'finishing', 'heading_accuracy',
       'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy',
       'long_passing', 'ball_control', 'acceleration', 'sprint_speed',
       'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'stamina',
       'strength', 'long_shots', 'aggression', 'interceptions', 'positioning',
       'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle',
       'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning',
       'gk_reflexes'],
      dtype='object')
```

In [53]:
```python
x = player_attr_df[['gk_handling']].values
y = player_attr_df[['gk_reflexes']].values
```

Split the data into training and testing in 0.3 ratio (70% training, 30% testing)

In [54]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3)
```

In [55]:
```python
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

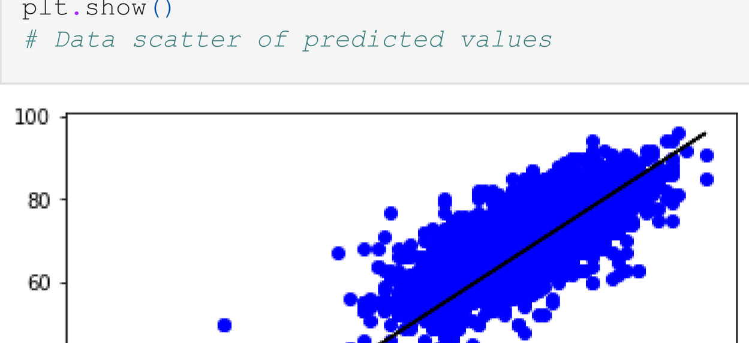
```
(128784, 1) (55194, 1) (128784, 1) (55194, 1)
```

Linear Regression

In [56]:
```python
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(X_train, Y_train)
print(lm.score(X_test, Y_test))
```

```
0.9342753939892032
```

In [57]:
```python
y_pred = lm.predict(X_test)
plt.scatter(X_test, Y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')

plt.show()
# Data scatter of predicted values
```



In [58]:
```python
Y_predicted = lm.predict(X_test) # Input Code Here
rsquared = r2_score(Y_predicted, Y_test)
print("R2 Score: " + str(rsquared))
```

```
R2 Score: 0.9293650051144279
```

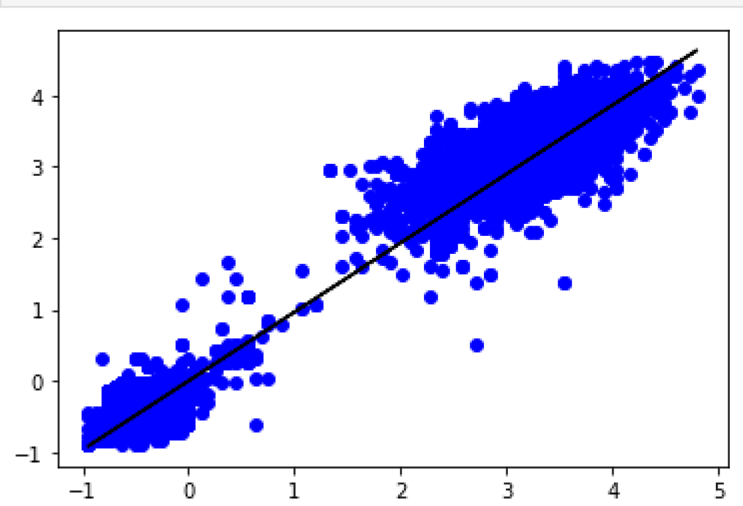
Standar Scalar transformation

In [59]:
```python
from sklearn.preprocessing import StandardScaler
# define standard scaler
sc = StandardScaler()

x = sc.fit_transform(x) # Input Code Here
y = sc.fit_transform(y) # Input Code Here
```

In [60]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3)

lm = LinearRegression()
lm.fit(X_train, Y_train)
print(lm.score(X_test, Y_test))
```

```
0.9337446077054987
```

In [61]:
```python
y_pred = lm.predict(X_test)
plt.scatter(X_test, Y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')

plt.show()
# Data scatter of predicted values
```



In [62]:
```python
Y_predicted = lm.predict(X_test) # Input Code Here
rsquared = r2_score(Y_predicted, Y_test)
print("R2 Score: " + str(rsquared))
```

```
R2 Score: 0.928781356085286
```

In [63]:
```python
lm = LinearRegression()
scores = cross_val_score(lm, X_train, Y_train, scoring='r2', cv=5)
scores
```

Out[63]:
```
array([0.93292487, 0.93100319, 0.93453493, 0.93633306, 0.93421865])
```

In [64]:
```python
# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, Y_train, scoring='r2', cv=folds)
print(scores.mean())
```

```
0.9338093939317818
```

In [74]:
```python
def evaluate_model(Model, lambdas):
    training_errors = [] # we will store the error on the training set, for using each different lambda
    testing_errors = [] # and the error on the testing set
    for l in lambdas:
        # in sklearn, they refer to lambda as alpha, the name is different in different literature
        # Model will be either Lasso, Ridge or ElasticNet
        model = Model(alpha=l, max_iter=1000) # we allow max number of iterations until the model converges
        model.fit(X_train, Y_train)

        training_predictions = model.predict(X_train)
        training_mse = mean_squared_error(Y_train, training_predictions)
        training_errors.append(training_mse)

        testing_predictions = model.predict(X_test)
        testing_mse = mean_squared_error(Y_test, testing_predictions)
        testing_errors.append(testing_mse)
    return training_errors, testing_errors
```

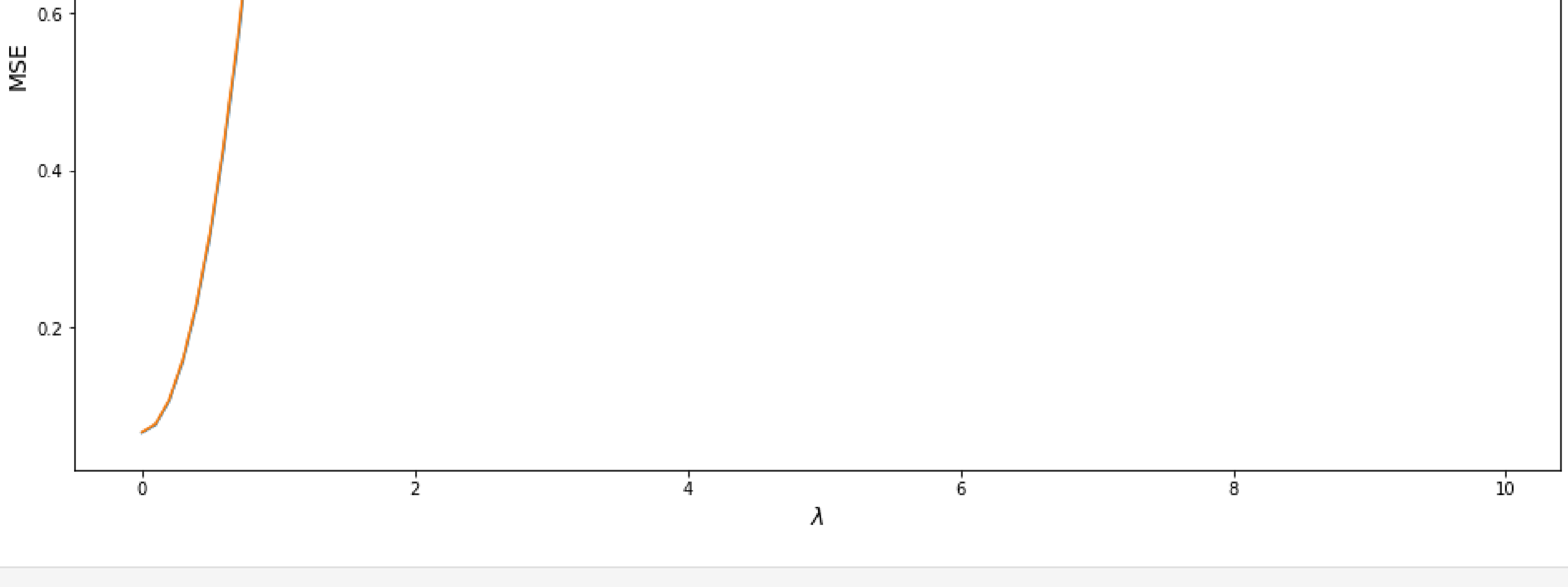
In [75]:
```python
def plot_errors(lambdas, train_errors, test_errors, title):
    plt.figure(figsize=(16, 9))
    plt.plot(lambdas, train_errors, label="train")
    plt.plot(lambdas, test_errors, label="test")
    plt.xlabel("$\\lambda$", fontsize=14)
    plt.ylabel("MSE", fontsize=14)
    plt.title(title, fontsize=20)
    plt.legend(fontsize=14)
    plt.show()
```

In [66]:
```python
# import lasso regression library
from sklearn.linear_model import Lasso

# using lasso to fit X_train, y_train with 10-fold cross-validation over alpha values from 0.1 to 0.9
for i in range (1,10):
    lasso = Lasso(alpha=i/10)
    scores = cross_val_score(lasso, X_train, Y_train, cv=10)
    print(i/10, ": ", scores.mean())
```

```
0.1 :  0.923737329196698
0.2 :  0.8935916704786845
0.3 :  0.843343567095755
0.4 :  0.77299301904791
0.5 :  0.6825400263351492
0.6 :  0.571984588957473
0.7 :  0.44132670691488085
0.8 :  0.29056638020737313
0.9 :  0.11970360883494972
```

In [76]:
```python
from sklearn.linear_model import ElasticNet, Lasso, Ridge
from sklearn.metrics import mean_squared_error  # we will use MSE for evaluation
import matplotlib.pyplot as plt
```

In [77]:
```python
import warnings
warnings.filterwarnings('ignore')
# let's generate different values for lambda from 0 (no-regularization) and (10 too much regularization)
lambdas = np.arange(0, 10, step=0.1)

lasso_train, lasso_test = evaluate_model(Lasso, lambdas)
plot_errors(lambdas, lasso_train, lasso_test, "Lasso")
```

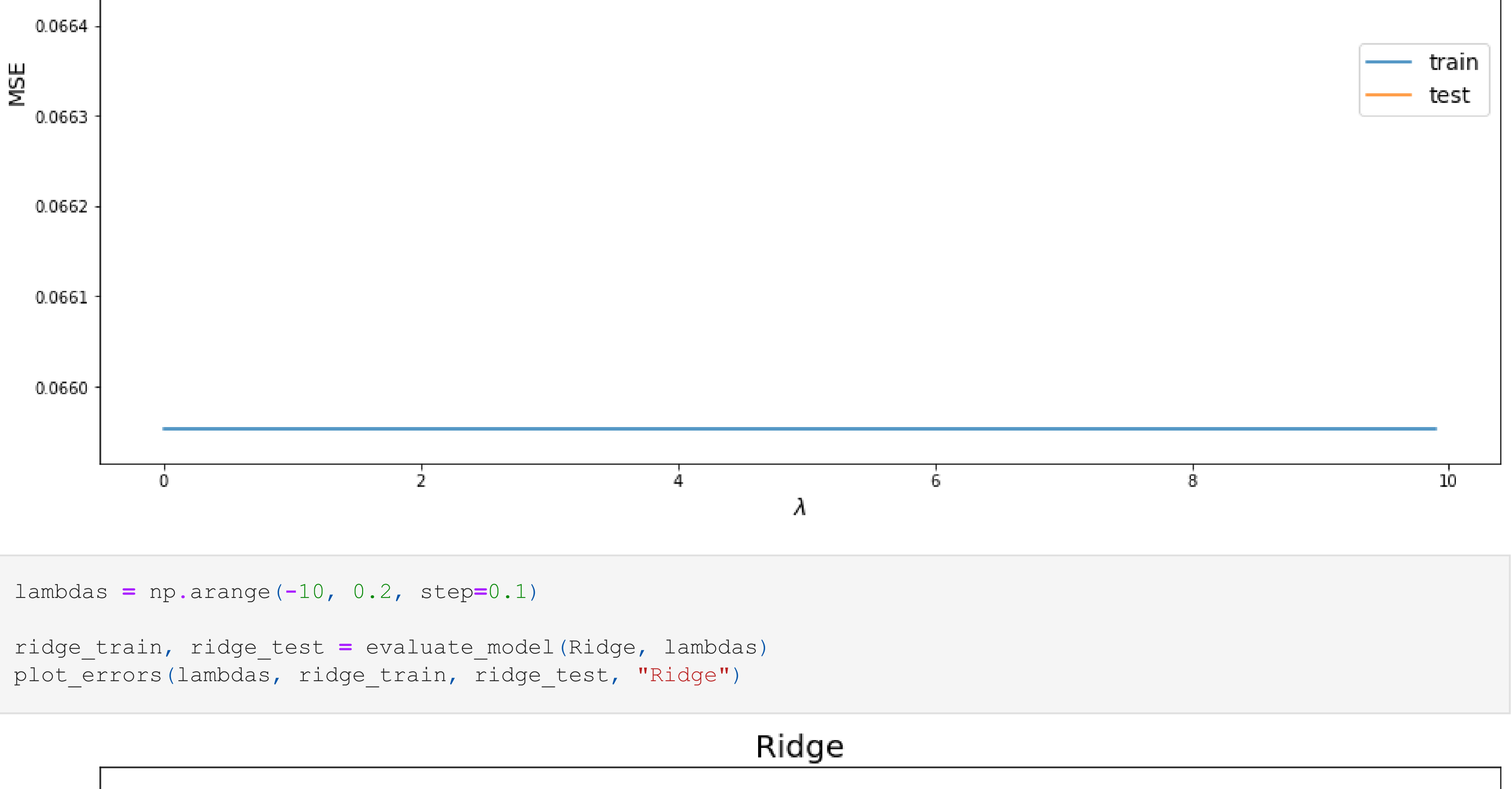


In [78]:
```python
lambdas = np.arange(-10, 0.2, step=0.1)

lasso_train, lasso_test = evaluate_model(Lasso, lambdas)
plot_errors(lambdas, lasso_train, lasso_test, "Lasso")
```

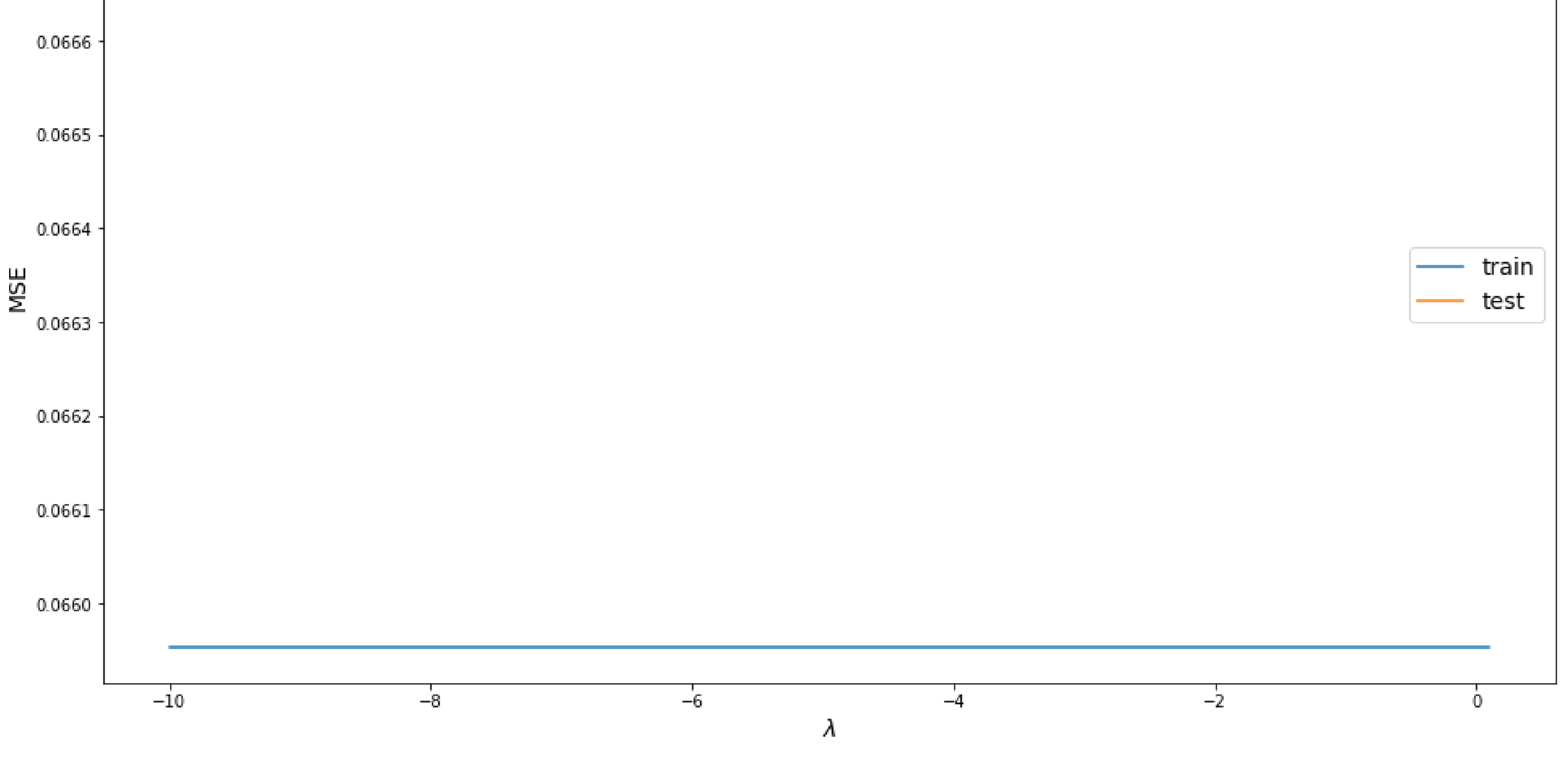
In [29]:
```python
# import ridge regression library
from sklearn.linear_model import Ridge

# using ridge to fit X_train, y_train with 10-fold cross-validation over alpha values from 0.1 to 0.9
for i in range (1,10):
    ridge = Ridge(alpha=i/10)
    scores = cross_val_score(ridge, X_train, Y_train, cv=10)
    print(i/10, ": ", scores.mean())
```

```
0.1 :  0.9344291402583
0.2 :  0.9344291402989555
0.3 :  0.9344291403382494
0.4 :  0.9344291403761817
0.5 :  0.9344291404127528
0.6 :  0.9344291404479625
0.7 :  0.9344291404818105
0.8 :  0.9344291405142974
0.9 :  0.9344291405454225
```

In [79]:
```python
# let's generate different values for lambda from 0 (no-regularization) and (10 too much regularization)
lambdas = np.arange(0, 10, step=0.1)

ridge_train, ridge_test = evaluate_model(Ridge, lambdas)
plot_errors(lambdas, ridge_train, ridge_test, "Ridge")
```



In [81]:
```python
lambdas = np.arange(-10, 0.2, step=0.1)

ridge_train, ridge_test = evaluate_model(Ridge, lambdas)
plot_errors(lambdas, ridge_train, ridge_test, "Ridge")
```



In [ ]: