1) Only Final Data Structures

A data structure is a specialized format for organizing, processing, retrieving and storing data. For example, here we use a dictionary to store user input which is then stored inside a variable.

```
#Get the feature input from the user
def get_user_input():
    pregnancies = st.sidebar.slider('pregnancies', 0, 17, 3)
    glucose = st.sidebar.slider('glucose', 0, 119, 117)
    blood_pressure = st.sidebar.slider('blood_pressure', 0, 122, 72)
    skin_thickness = st.sidebar.slider('skin_thickness', 0, 99, 23)
    insulin = st.sidebar.slider('insulin', 0.0, 846.0, 30.0)
    BMI = st.sidebar.slider('BMI', 0.0, 67.1, 32.0)
    DPF = st.sidebar.slider('DPF', 0.078, 2.42, 0.3725)
    age = st.sidebar.slider('age', 21, 81, 29)

    #Store a dictionary into a variable
    user_data = {'pregnancies' : pregnancies,
                 'glucose' : glucose,
                 'blood_pressure' : blood_pressure,
                 'skin_thickness' : skin_thickness,
                 'insulin' : insulin,
                 'BMI' : BMI,
                 'DPF' : DPF,
                 'age' : age,
                 }

    #Transform the data into a data frame
    features = pd.DataFrame(user_data, index = [0])
    return features

#Store the user_input into a variable
user_input = get_user_input()
```

2) Side Effect Free Functions (Also known as Pure Function)

Functions that are **deterministic** and don't have **side effects**, are called "pure functions". For example, here the user input (get_user_input) is stored inside a variable (user_input) and is passed through the model. These inputs doesn't get changed and remains the same throughout the execution of the code (ie. not affected by anything outside of the function and doesn't affect anything outside of the function)

```
#Get the feature input from the user
def get_user_input():
    pregnancies = st.sidebar.slider('pregnancies', 0, 17, 3)
    glucose = st.sidebar.slider('glucose', 0, 119, 117)
    blood_pressure = st.sidebar.slider('blood_pressure', 0, 122, 72)
    skin_thickness = st.sidebar.slider('skin_thickness', 0, 99, 23)
    insulin = st.sidebar.slider('insulin', 0.0, 846.0, 30.0)
    BMI = st.sidebar.slider('BMI', 0.0, 67.1, 32.0)
    DPF = st.sidebar.slider('DPF', 0.078, 2.42, 0.3725)
    age = st.sidebar.slider('age', 21, 81, 29)

    #Store a dictionary into a variable
    user_data = {'pregnancies' : pregnancies,
                 'glucose' : glucose,
                 'blood_pressure' : blood_pressure,
                 'skin_thickness' : skin_thickness,
                 'insulin' : insulin,
                 'BMI' : BMI,
                 'DPF' : DPF,
                 'age' : age,
                 }

    #Transform the data into a data frame
    features = pd.DataFrame(user_data, index = [0])
    return features

#Store the user_input into a variable
user_input = get_user_input()
```

3) Higher Order Function (HOF)

Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions. For example, hist_observation and heatmap_observation functions are called by the init function, which takes in all the parameters required for these functions to run.

```
class Plot:

    def __init__(self, data, path):
        self.data = data
        self.path = path
        self.hist_observation()
        self.heatmap_observation()

    def hist_observation(self):
        fig, ax = plt.subplots()
        self.data[['Insulin', 'Glucose', 'BloodPressure',]].hist(bins=30, figsize=(1500,1000), ax=ax)
        fig.savefig("{}/histogram.png".format(self.path))

    def heatmap_observation(self):
        fig, ax = plt.subplots()
        d = self.data[['Pregnancies', 'Glucose', 'BloodPressure',
       'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']]
        dataplot = sns.heatmap(d.corr(), annot=True, cmap="YlGnBu")
        fig.savefig("{}/heatmap.png".format(self.path))
```

4) Functions as Parameters and Return Values

These are functions that takes in input as parameters and gives output based on return statement. For example, here inside the init method we call split, train, predict method and return a value of training split (X_train, y_train), test split (X_test, Y_test), get accuracy score of training data, get mean squared error value on y_test and y_pred. we take in the parameters as df (our dataframe) , X_train, X_test, y_train and y_test and return values from these functions.

```
class ModelBuilding:

    def __init__(self, df, path=None, X_train=None, X_test=None,
    y_train=None, y_test=None):
        self.data = df
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test
        self.y_pred = None
        Plot(df, path=path)
        self.split()
        self.train()
        self.predict()

    def split(self):
        X = self.data[['Pregnancies', 'Glucose', 'BloodPressure',
       'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']]
        y = self.data['Outcome']
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y,
        test_size=0.3)
        return self.X_train, self.X_test, self.y_train, self.y_test

    def train(self):
        self.rcb = RandomForestClassifier()
        self.rcb.fit(self.X_train, self.y_train)
        score = self.rcb.score(self.X_train, self.y_train)
        return score


    def predict(self):
        self.y_pred = self.rcb.predict(self.X_test)
        mse = mean_squared_error(self.y_test, self.y_pred)
        return mse
```

5) Closure

Closure is a nested function that helps us access the outer function's variables even after the outer function is closed. For example, here we call the function get_user_input() from outside. It is assigned to a variable and is called at st.write(user_input) which is outside of the function. st.write works like print function in streamlit.

```
#Get the feature input from the user
def get_user_input():
    pregnancies = st.sidebar.slider('pregnancies', 0, 17, 3)
    glucose = st.sidebar.slider('glucose', 0, 119, 117)
    blood_pressure = st.sidebar.slider('blood_pressure', 0, 122, 72)
    skin_thickness = st.sidebar.slider('skin_thickness', 0, 99, 23)
    insulin = st.sidebar.slider('insulin', 0.0, 846.0, 30.0)
    BMI = st.sidebar.slider('BMI', 0.0, 67.1, 32.0)
    DPF = st.sidebar.slider('DPF', 0.078, 2.42, 0.3725)
    age = st.sidebar.slider('age', 21, 81, 29)

    #Store a dictionary into a variable
    user_data = {'pregnancies' : pregnancies,
                 'glucose' : glucose,
                 'blood_pressure' : blood_pressure,
                 'skin_thickness' : skin_thickness,
                 'insulin' : insulin,
                 'BMI' : BMI,
                 'DPF' : DPF,
                 'age' : age,
                 }

    #Transform the data into a data frame
    features = pd.DataFrame(user_data, index = [0])
    return features

#Store the user_input into a variable
user_input = get_user_input()

#Set a subheader and display the users input
st.subheader('User input:')
st.write(user_input)
```