# Прогнозирование временных рядов.

In [1]:

```
# Импорт библиотек
import numpy as np
import pandas as pd

# Импорт matplotlib
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
%matplotlib inline
```

**Данные о пассажирских авиаперевозках:**

Международные пассажирские авиаперевозки в тысячах человек за каждый месяц с января 1949 по декабрь 1960 года.

In [2]:

```
ser_g = pd.read_csv('series_g.csv', sep=';')
ser_g.head()
```

Out[2]:

| | date | series_g |
|---|---|---|
| 0 | JAN 1949 | 112 |
| 1 | FEB 1949 | 118 |
| 2 | MAR 1949 | 132 |
| 3 | APR 1949 | 129 |
| 4 | MAY 1949 | 121 |

In [3]:

```
# Преобразуем строчки с датами в объект datetime
# format='%b %Y' означает, что в нашей строчке сначала идёт трёхбуквенное название месяца (
ser_g['date'] = pd.to_datetime(ser_g['date'], format='%b %Y')
#  Добавляем столбоц с логарифмом от объема пассажироперевозок
ser_g['log_y'] = np.log10(ser_g['series_g'])
```

In [4]:

```
#  Смотрим, что получилось
ser_g.head()
```

Out[4]:

| | date | series_g | log_y |
|---|---|---|---|
| 0 | 1949-01-01 | 112 | 2.049218 |
| 1 | 1949-02-01 | 118 | 2.071882 |
| 2 | 1949-03-01 | 132 | 2.120574 |
| 3 | 1949-04-01 | 129 | 2.110590 |
| 4 | 1949-05-01 | 121 | 2.082785 |

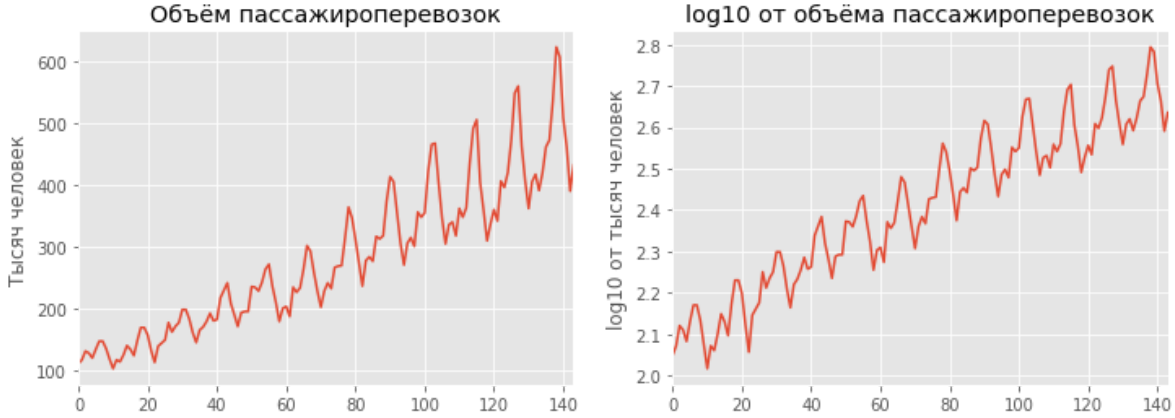
Построим графики объёма пассажироперевозок и проверим, какой тип тренда (линейный или нет) и какой тип сезонности ( аддитивный или мультипликативный), наблюдается. По первому графику уже прослеживается линейный тренд и мультипликативная сезонность. Но чтобы окончательно убедиться в последнем, добавим график логарифма от этой же величины. После логирафмирования циклы стали одинаковой высоты, а это и говорит о мультипликативном харакрете сезонности.

In [5]:

```
fig = plt.figure(figsize=(12, 4))

ax1 = fig.add_subplot(121)
ser_g['series_g'].plot(ax=ax1)
ax1.set_title(u'Объём пассажироперевозок')
ax1.set_ylabel(u'Тысяч человек')

ax2 = fig.add_subplot(122)
pd.Series(ser_g['log_y']).plot(ax=ax2)
ax2.set_title(u'log10 от объёма пассажироперевозок')
ax2.set_ylabel(u'log10 от тысяч человек')
```

Out[5]:

```
Text(0, 0.5, 'log10 от тысяч человек')
```




# Прогнозирование с использованием линейной регрессии

Будем строить модель линейной регрессии для приближения логарифма от объёма перевозок. То есть

$$\log y_i = \beta x_i + c(x_i) + \varepsilon_i,$$

где $y_i$ -- объём перевозок, $x_i$ -- порядковый номер месяца, $c(x_i)$ -- сезонная составляющая, $\varepsilon_i$ -- случайный шум.

Для удобства дальнейшего использования создадим дополнительно 12 новых месяцев для построения прогноза на них. Для этого создадим эти 12 новых дат с помощью функции `pd.date_range`. Данный объект будет объектом класса `DateTimeIndex` (наследованный от класса `Index`), и чтобы объединить их с колонкой `ser_g['date']`, принадлежащей классу `datetime64`, придётся привести последнюю к классу `Index`. Объединим два набора дат и сохраним их в объекте `new_dates`.

Далее создадим фиктивный датафрейм `df2`, состоящий из одной колонки с этими новыми датами, и приклеим его к исходному датафрейму `ser_g` с помозью функции `pd.merge`, создав при этом датафрейм `df`. Эта функция склеивает два датасета по указанному набору колонок (параметр `on`) и по указанному правилу склейки (параметр `how`). В `on` указываем одну общую колонку `date`, по которой нужно произвести склейку. В `how` указываем `right`, что означает следующее: возьми весь правый датасет и приклей к нему левый датасет по условию совпадения значений колонки `on`, а в случае если для значений из правой колонки `on` не найдётся соотвествующих значений в левой колонке `on`, то тогда приклей `NaN` значения. Вообще говоря, опция `how` соответствует опциям `JOIN` в языке SQL (`LEFT JOIN`, `RIGHT_JOIN`, `INNER JOIN`, `OUTER_JOIN`).

In [6]:

```
# Создаём последовательсть месяцев. freq='MS' означает первое число каждого месяца из указа
new_dates = pd.date_range('1961-01-01', '1961-12-01', freq='MS')

# Приводим ser_g['date'] к типу Index, объединяем с 12 месяцами, полученными на предыдущем
new_dates = pd.Index(ser_g['date']) | new_dates

# Создаём датафрейм из одной колонки с расширенным набором дат
df2 = pd.DataFrame({'date': new_dates})

# Объединяем два датафрейма по колонке 'date' в датафрейм df.
# опция how соответствует опциям JOIN в SQL
df = pd.merge(ser_g, df2, on='date', how='right')
```

Создадим регрессионную переменную `month_num` -- порядковый номер пары (месяц, год). И прологарифмируем таргет.

In [7]:

```
df['month_num'] = range(1, len(df) + 1)
df['log_y'] = np.log10(df['series_g'])
```

Создадим 12 колонок `season_1`, `season_2`, ..., `season_12`, в которые поместим индикаторы соответствующего месяца. Чтобы достать порядковый номер месяца в каждой строчке, применим последовательно пару методов `dt` и `month` к колонке `df['date']`. Внутри цикла будем проверять, равен ли очередной месяц текущему значению из цикла.

In [8]:

```
for x in range(1, 13):
    df['season_' + str(x)] = df['date'].dt.month == x
```

Правда, для устранения линейной зависимости между колонками, один из сезонных индикаторов придётся исключить. Пусть базовым месяцем будет январь.

In [9]:

```
# xrange(2, 13) соответствует всем месяцам с февраля по декабрь
season_columns = ['season_' + str(x) for x in range(2, 13)]

# Создадим объекты матрицу X и вектор y для обучения модели
X = df[['month_num'] + season_columns]
y = df['log_y']

# Оставим только те строчки, у которых известны значения y (с номерами от 60 до 144).
# Первые 60 наблюдений отбрасываем, потому, что они испортят прогноз.
X1 = X[60:144]
y1 = y[60:144]
```

Настраиваем линейную регрессионную модель.

In [10]:

```
from sklearn.linear_model import LinearRegression
```

In [11]:

```
model = LinearRegression()
model.fit(X1, y1)
```

Out[11]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
se)
```

Считаем качество модели (коэффициент $R^2$).

In [12]:

```
print ('R^2: {0}'.format(model.score(X1, y1)))
```

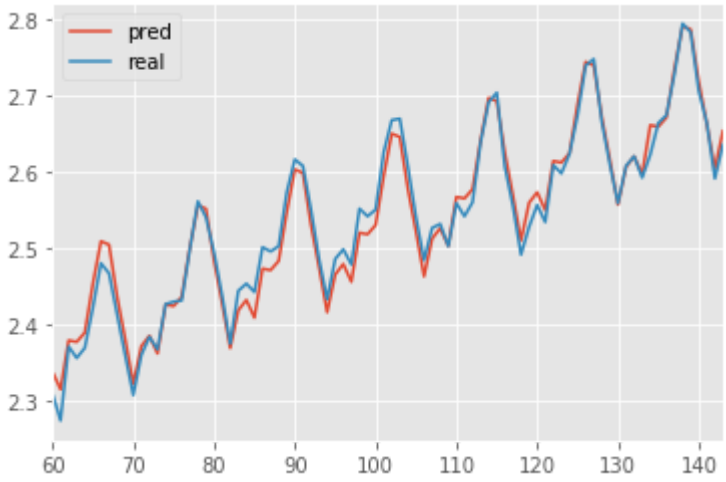
```
R^2: 0.9748793547553932
```

In [13]:

```
pred = pd.DataFrame({
    'pred': model.predict(X1),
    'real': y1})
pred.plot()
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b61371bfd0>
```



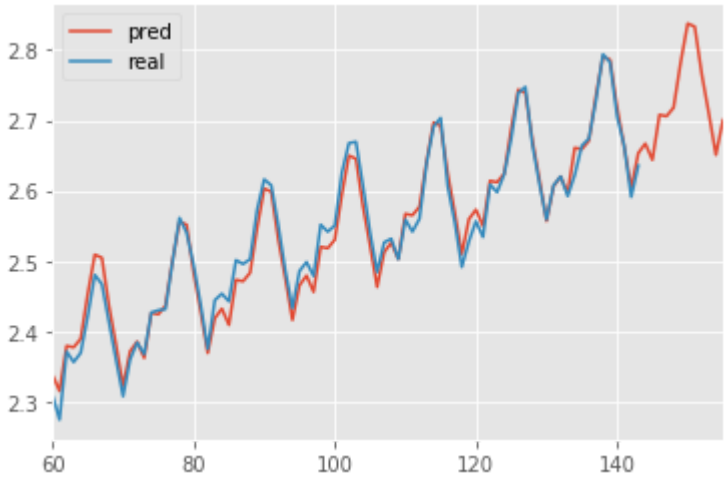

Теперь построим предсказание для всей матрицы X , включая неизвестные добавленных 12 месяцев.

In [14]:

```
pred = pd.DataFrame({
    'pred': model.predict(X[60:]),
    'real': y[60:]})
pred.plot()
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b6137b2438>
```




Выведем предсказанные значения на 12 месяцев вперед (тыс.чел. в месяц)

In [15]:

```
10**model.predict(X[144:])
```

Out[15]:

```
array([464.32718369, 440.26033056, 510.22193527, 507.93449309,
       522.24325509, 605.27326499, 687.95961997, 680.97578391,
       582.42250084, 512.32243127, 447.48144967, 501.39937568])
```

# Прогнозирование с использованием нейронной сети

In [16]:

```
# Смотрим на данные
ser_g.head()
```

Out[16]:

| | date | series_g | log_y |
|---|---|---|---|
| 0 | 1949-01-01 | 112 | 2.049218 |
| 1 | 1949-02-01 | 118 | 2.071882 |
| 2 | 1949-03-01 | 132 | 2.120574 |
| 3 | 1949-04-01 | 129 | 2.110590 |
| 4 | 1949-05-01 | 121 | 2.082785 |

In [17]:

```
# Преобразуем данные

# Создаем таблицу таким образом, что бы каждая строка сообветствовала периоду (1 год):
# в строке 13 (12+1) значений, соответствующих показателю за каждый месяц,
# значение за 13 месяц будет как-бы предсказанием по предыдущим 12 (вектор Y, для обучающей

ser_g_2 = pd.DataFrame()

for i in range(12,0,-1):
    ser_g_2['t-'+str(i)] = ser_g.iloc[:,2].shift(i)

ser_g_2['t'] = ser_g.iloc[:,2].values

print(ser_g_2.head(13))
```

```
        t-12      t-11      t-10       t-9       t-8       t-7       t-6  \
0        NaN       NaN       NaN       NaN       NaN       NaN       NaN
1        NaN       NaN       NaN       NaN       NaN       NaN       NaN
2        NaN       NaN       NaN       NaN       NaN       NaN       NaN
3        NaN       NaN       NaN       NaN       NaN       NaN       NaN
4        NaN       NaN       NaN       NaN       NaN       NaN       NaN
5        NaN       NaN       NaN       NaN       NaN       NaN       NaN
6        NaN       NaN       NaN       NaN       NaN       NaN  2.049218
7        NaN       NaN       NaN       NaN       NaN  2.049218  2.071882
8        NaN       NaN       NaN       NaN  2.049218  2.071882  2.120574
9        NaN       NaN       NaN  2.049218  2.071882  2.120574  2.110590
10       NaN       NaN  2.049218  2.071882  2.120574  2.110590  2.082785
11       NaN  2.049218  2.071882  2.120574  2.110590  2.082785  2.130334
12  2.049218  2.071882  2.120574  2.110590  2.082785  2.130334  2.170262

         t-5       t-4       t-3       t-2       t-1         t
0        NaN       NaN       NaN       NaN       NaN  2.049218
1        NaN       NaN       NaN       NaN  2.049218  2.071882
2        NaN       NaN       NaN  2.049218  2.071882  2.120574
3        NaN       NaN  2.049218  2.071882  2.120574  2.110590
4        NaN  2.049218  2.071882  2.120574  2.110590  2.082785
5   2.049218  2.071882  2.120574  2.110590  2.082785  2.130334
6   2.071882  2.120574  2.110590  2.082785  2.130334  2.170262
7   2.120574  2.110590  2.082785  2.130334  2.170262  2.170262
8   2.110590  2.082785  2.130334  2.170262  2.170262  2.133539
9   2.082785  2.130334  2.170262  2.170262  2.133539  2.075547
10  2.130334  2.170262  2.170262  2.133539  2.075547  2.017033
11  2.170262  2.170262  2.133539  2.075547  2.017033  2.071882
12  2.170262  2.133539  2.075547  2.017033  2.071882  2.060698
```

In [18]:

```
#  Отрезаем первые 12 строк (строки с NaN)
ser_g_4 = ser_g_2[12:]
#  Проверяем, что все в порядке
ser_g_4.head()
```

Out[18]:

| | t-12 | t-11 | t-10 | t-9 | t-8 | t-7 | t-6 | t-5 | t-4 |
|---|---|---|---|---|---|---|---|---|---|
| 12 | 2.049218 | 2.071882 | 2.120574 | 2.110590 | 2.082785 | 2.130334 | 2.170262 | 2.170262 | 2.133539 |
| 13 | 2.071882 | 2.120574 | 2.110590 | 2.082785 | 2.130334 | 2.170262 | 2.170262 | 2.133539 | 2.075547 |
| 14 | 2.120574 | 2.110590 | 2.082785 | 2.130334 | 2.170262 | 2.170262 | 2.133539 | 2.075547 | 2.017033 |
| 15 | 2.110590 | 2.082785 | 2.130334 | 2.170262 | 2.170262 | 2.133539 | 2.075547 | 2.017033 | 2.071882 |
| 16 | 2.082785 | 2.130334 | 2.170262 | 2.170262 | 2.133539 | 2.075547 | 2.017033 | 2.071882 | 2.060698 |

In [19]:

```
print(ser_g_4.shape)
```

```
(132, 13)
```

In [20]:

```
#  предикторы и отклик разделяем
# Отклик  вектор y
y = ser_g_4['t']
# Предикторы - таблица X
X = ser_g_4.drop('t', axis=1)
```

In [21]:

```
# Разделяем на обучающую и тестовую выборки
# В обучающую первые 60 значений не включаем, что бы не испортить прогноз
# Тестовая - последние наблюдения
X_train = X[60:120]
y_train = y[60:120]
X_test  = X[120:]
y_test  = y[120:]
```

In [22]:

```
#  Все хорошо?
print(ser_g_4.shape)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(132, 13)
(60, 12)
(60,)
(12, 12)
(12,)
```

In [23]:

```
#  Преобразование pandas dataframe в numpy array
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values
```

In [24]:

```
# Подключаем библиотеки для создания нейронной сети
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.initializers import RandomNormal, RandomUniform
from tensorflow.keras import optimizers
```

In [25]:

```
# Создание модели
model = Sequential()
model.add(Dense(6, input_dim=12, activation='relu'))
model.add(Dense(1, activation='linear'))

# Компиляция
#sgd = SGD(learning_rate=0.01, momentum=0.0, nesterov=False)
#adanm = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_percenta
```

In [26]:

```
# Обучение нейронной сети

model.fit(X_train, y_train, epochs=10000)
```

```
4 - mean_absolute_percentage_error: 0.5245
Epoch 9982/10000
60/60 [==============================] - 0s 116us/sample - loss: 3.5547e-0
4 - mean_absolute_percentage_error: 0.5647
Epoch 9983/10000
60/60 [==============================] - 0s 166us/sample - loss: 3.2096e-0
4 - mean_absolute_percentage_error: 0.5367
Epoch 9984/10000
60/60 [==============================] - 0s 150us/sample - loss: 3.2711e-0
4 - mean_absolute_percentage_error: 0.5326
Epoch 9985/10000
60/60 [==============================] - 0s 116us/sample - loss: 3.2225e-0
4 - mean_absolute_percentage_error: 0.5439
Epoch 9986/10000
60/60 [==============================] - 0s 166us/sample - loss: 3.2570e-0
4 - mean_absolute_percentage_error: 0.5392
Epoch 9987/10000
60/60 [==============================] - 0s 116us/sample - loss: 3.1963e-0
4 - mean_absolute_percentage_error: 0.5378
Epoch 9988/10000
```

In [27]:

```
# оценка качества модели на тестовом множестве
scores = model.evaluate(X_test, y_test)
print("\nMAPE: %.2f%%" % (scores[1]))
```

```
12/1 [==============================================================================
==============================================================================
==============================================================================
==============================================================================
==========================================================] - 0s 7ms/sam
ple - loss: 5.4461e-04 - mean_absolute_percentage_error: 0.6699

MAPE: 0.67%
```

In [28]:

```
# Вычисляем прогноз
predictions = model.predict(X_test)
# round predictions
# rounded = [round(x[0]) for x in predictions]
# print(rounded)
```

In [29]:

```
# Вычисляем подгонку
predictions_train = model.predict(X_train)
```

In [30]:

```
# Вспоминаем размеры таблиц
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60, 12)
(60,)
(12, 12)
(12,)
```

In [31]:

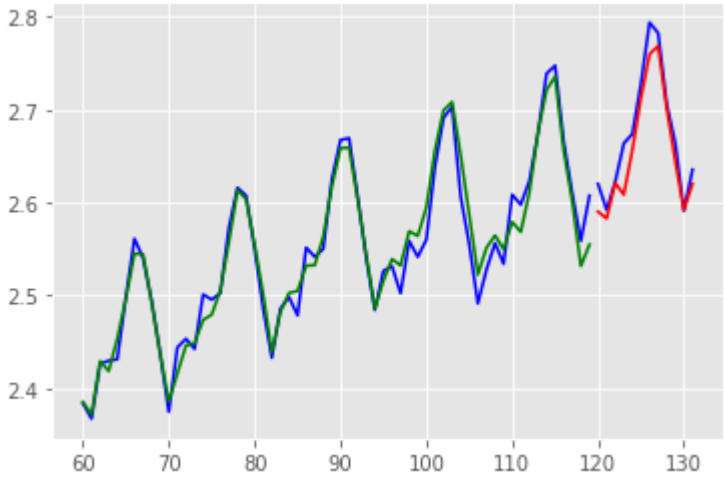
```
#   было 144 наблюдений

#   отбросили 12 стало 132
#   train 120
#   test  12
```

In [32]:

```
#   График с результатами

x2 = np.arange(60, 120, 1)
x3 = np.arange(120, 132, 1)

plt.plot(x2, y_train, color='blue')
plt.plot(x2, predictions_train, color='green')

plt.plot(x3, y_test, color='blue')
plt.plot(x3, predictions, color='red')
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x1b618d52eb8>]
```



In [33]:

```
x = np.roll((X_test[-1]).copy(), -1)
print(x)
print()

x[-1] = y_test[-1]

print(x)
```

```
[2.62013605 2.59217676 2.62221402 2.66370093 2.673942   2.72835378
 2.79379038 2.78247262 2.70586371 2.66370093 2.59106461 2.60745502]

[2.62013605 2.59217676 2.62221402 2.66370093 2.673942   2.72835378
 2.79379038 2.78247262 2.70586371 2.66370093 2.59106461 2.63548375]
```

In [34]:

```
# создаем массив с месяцами для предсказания

def create_in_val():
    x = np.roll((X_test[-1]).copy(), -1)
    x[-1] = y_test[-1]
    return x

def predict (n):
    x = create_in_val()
    predictions = np.zeros((n,1))

    for i in range(0, n):
        p = model.predict(np.array([x]))
        predictions[i] = p

        for j in range(0, 11):
            x[j] = x[j + 1]
        x[11] = p

    return predictions
```

Для того, что бы получить предсказания на будущий год необходимо сделать 24 предсказания, т.к. данные за последний год ушли на тестовую выборку

In [35]:

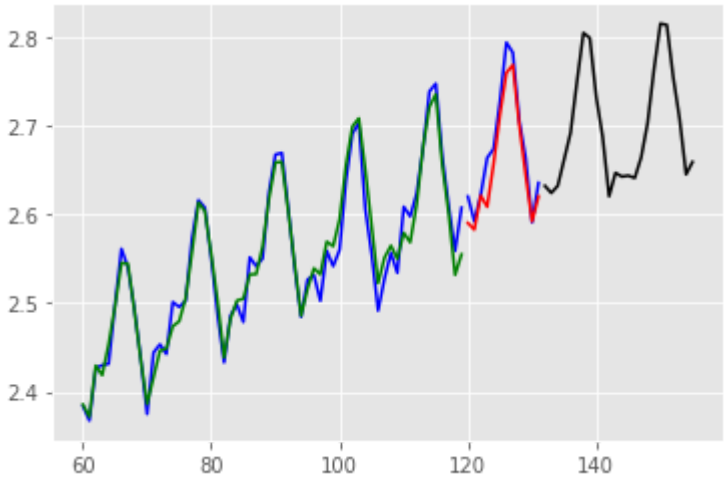
```
# n - количество предсказаний, которые будут сделаны с помощью модели
n = 24
predictions_future = predict(n)
```

In [36]:

```
x2 = np.arange(60, 120, 1)
x3 = np.arange(120, 132, 1)
x4 = np.arange(132, 132+n, 1)

plt.plot(x2, y_train, color='blue')
plt.plot(x2, predictions_train, color='green')

plt.plot(x3, y_test, color='blue')
plt.plot(x3, predictions, color='red')
plt.plot(x4, predictions_future, color='black')
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x1b61ab409b0>]
```



In [37]:

```
#смотрим на предсказанные на будущее значения (логарифмический масштаб)
predictions_future[n-12:]
```

Out[37]:

```
array([[2.64258289],
       [2.64364672],
       [2.64103651],
       [2.66425228],
       [2.70283461],
       [2.7635603 ],
       [2.81519198],
       [2.81371903],
       [2.75462842],
       [2.70732951],
       [2.64507771],
       [2.65905094]])
```

In [38]:

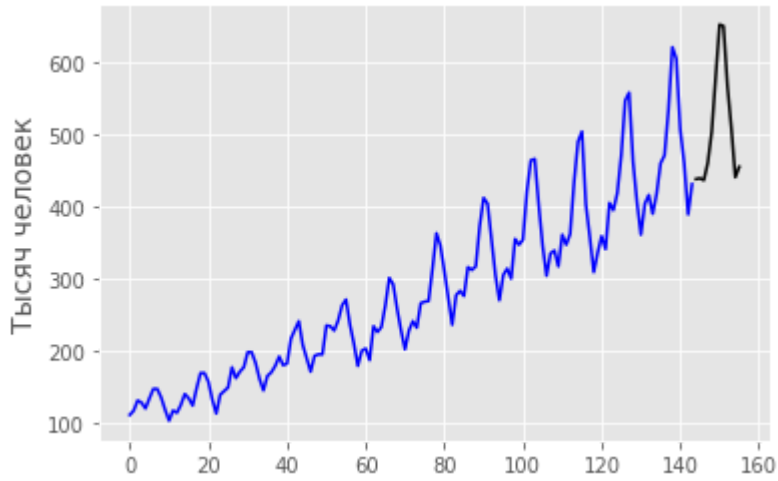
```
#переводим из логарифмического масштаба
predictions_future_2 = 10**predictions_future.copy()
```

In [39]:

```
x1 = np.arange(0, 144, 1)
x2 = np.arange(144, 144+12, 1)

plt.plot(x1, ser_g['series_g'], color='blue')
plt.plot(x2, predictions_future_2[n-12:], color='black')
plt.ylabel('Тысяч человек', fontsize=14)
```

Out[39]:

```
Text(0, 0.5, 'Тысяч человек')
```




Выведем предсказанные значения на 12 месяцев вперед (тыс.чел. в месяц):

In [40]:

```
predictions_future_2[n-12:]
```

Out[40]:

```
array([[439.11967199],
       [440.19663371],
       [437.55888851],
       [461.58563128],
       [504.46914229],
       [580.17671554],
       [653.4193393 ],
       [651.20696092],
       [568.36643136],
       [509.71746159],
       [441.64946164],
       [456.09041088]])
```