0.0.0 - база ООП

In [1]:
```
class TAnimal:
    name = ""

    def __init__(self, name):
        self.name = name
    def say(self):
        print(self.name)

Animal = TAnimal("Monkey")
Animal.say()
```

```
Monkey
```

In [2]:
```
class TCat(TAnimal):
    def may(self):
        print("May!")

Cat = TCat("Cat")
Cat.say()
Cat.may()
```

```
Cat
May!
```

In [ ]:
```
class TCat(TAnimal):
    def __init__(self):
        super().__init__("Cat")
    def may(self):
        print("May!")

Cat = TCat()
Cat.say()
Cat.may()
```

```
Cat
May!
```

In [4]:
```
class TDo:
    def Operation(self, x, y):
        return x + y
    def Run(self):
        x = int(input("Enter x > "))
        y = int(input("Enter y > "))
        z = self.Operation(x, y)
        print("Result = " + z.__str__())

Do = TDo()
Do.Run()

class TDo2(TDo):
    def Operation(self, x, y):
        return x * y
```

```
Result = 5
```

1.2.1 - Необходимо разработать виртуальную модель процесса обучения. В программе должны быть объекты-ученики, учитель, кладезь знаний. Потребуется

три класса – "учитель", "ученик", "данные". Учитель и ученик во многом похожи, оба – люди. Значит, их классы могут принадлежать одному надклассу "человек". Однако в контексте данной задачи у учителя и ученика вряд ли найдутся общие атрибуты. Определим, что должны уметь объекты для решения задачи "увеличить знания":

• Ученик должен уметь брать информацию и превращать ее в свои знания. • Учитель должен уметь учить группу учеников. • Данные могут представлять собой список знаний. Элементы будут извлекаться по индексу.

In [5]:
```
class Data:
    def __init__(self, *info):
        self.info = list(info)
    def __getitem__(self, i):
        return self.info[i]

class Teacher:
    def teach(self, info, *pupil):
        self.pupil = list(pupil)
        for i in pupil:
            i.take(info)

class Pupil:
    def __init__(self):
        self.knowledge = []
    def take(self, info):
        self.knowledge.append(info)

lesson = Data('class', 'object', 'inheritance', 'polymorphism', 'encapsulation')
marIvanna = Teacher()
vasy = Pupil()
pety = Pupil()
marIvanna.teach(lesson[2], vasy, pety)
marIvanna.teach(lesson[0], pety)
print(vasy.knowledge)
print(pety.knowledge)
```

```
['inheritance']
['inheritance', 'class']
```

1.2.2 - Напишите программу по следующему описанию. Есть класс "Воин". От него создаются два экземпляра-юнита. Каждому устанавливается здоровье в 100 очков. В случайном порядке они бьют друг друга. Тот, кто бьет, здоровья не теряет. У того, кого бьют, оно уменьшается на 20 очков от одного удара. После каждого удара надо выводить сообщение, какой юнит атаковал, и сколько у противника осталось здоровья. Как только у кого-то заканчивается ресурс здоровья, программа завершается сообщением о том, кто одержал победу.

In [7]:
```
import random



class Warrior:
    def __init__(self, health):
        self.health = health

    def hit(self, target, target1):
```

```
        if target.health > 0:
            target.health -= 20
        if target1 == warrior1:
            target1 = "Warrior1"
        if target1 == warrior2:
            target1 = "Warrior2"
        print(target1, " has attacked")
        print(target.health, " left")
        if target.health == 0:
            print(target1, " has won")



warrior1 = Warrior(100)
warrior2 = Warrior(100)
q = int(input("Enter 1 to attack. Enter 2 to stop program: "))


while q != 2:
    if q == 1:
        j = random.randint(1,3)
        if j % 2 == 0:
            warrior1.hit(warrior2, warrior1)
            q = int(input("Enter 1 to let some warrior attack: "))
        else:
            warrior2.hit(warrior2, warrior1)
            q = int(input("Enter 1 to let some warrior attack: "))
    else:
        print("Wrong input.")
        break
```

```
Warrior1  has attacked
80  left
Warrior1  has attacked
60  left
Warrior1  has attacked
40  left
Warrior1  has attacked
20  left
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
Warrior1  has attacked
0  left
Warrior1  has won
```

1.2.3 - Создайте класс по работе с дробями. В классе должна быть реализована следующая функциональность:

- сложение дробей;
- вычитание дробей;
- умножение дробей;
- деление дробей.

In [ ]:
```python
class Rational:

    @staticmethod
    def gcd(a,b):
        while (b != 0):
            (a, b) = (b, a % b)
        return a

    @staticmethod
    def sgn(x):
        if x > 0:
            return 1
        elif x < 0:
            return -1
        else:
            return 0
```

```
    def __init__(self, n, d):
        if n == 0:
            self.num = 0
            self.den = 1
        else:
            z = self.sgn(n)*self.sgn(d)
            n = abs(n)
            d = abs(d)
            k = self.gcd(n, d)
            self.num = z * n//k
            self.den = d//k

    def __str__(self):
        if self.num == 0:
            return "0"
        else:
            return str(self.num) + "/" + str(self.den)

    def __add__(self, o):
        n1 = self.num
        d1 = self.den
        if type(o) == int:
            n2 = o
            d2 = 1
        else:
            n2 = o.num
            d2 = o.den
        n = n1*d2 + n2*d1
        d = d1*d2
        return Rational(n, d)

    def __radd__(self, o):
        n1 = self.num
        d1 = self.den
        if type(o) == int:
            n2 = o
            d2 = 1
        else:
            n2 = o.num
            d2 = o.den
        n = n1*d2 + n2*d1
        d = d1*d2
        return Rational(n, d)

    def __sub__(self, o):
        n1 = self.num
        d1 = self.den
        n2 = o.num
        d2 = o.den
        n = n1*d2 - n2*d1
        d = d1*d2
        return Rational(n, d)

    def __mul__(self, o):
        n1 = self.num
        d1 = self.den
        n2 = o.num
        d2 = o.den
        n = n1*n2
        d = d1*d2
```

```
        return Rational(n, d)

    def __floordiv__(self, o):
        n1 = self.num
        d1 = self.den
        n2 = o.num
        d2 = o.den
        n = n1*d2
        d = d1*n2
        return Rational(n, d)


d1 = Rational(1, 2)
d2 = Rational(1, 3)
d3 = d1 + d2
print(d3)
d4 = d1 - d2
print(d4)
d5 = d1*d2
print(d5)
d6 = d1*d2
print(d6)
d7 = d1//d2
print(d7)
d8 = 6+d1
print(d8)
```

```
5/6
1/6
1/6
1/6
3/2
13/2
```

Задание - Создайте класс по работе с тригонометрическими функциями. В классе должны быть реализованы функции вычисления:

- косинуса;
- синуса;
- тангенса;
- арксинуса;
- арккосинуса;
- арктангенса;
- перевода из градусов в радианы.

In [ ]:
```
import math as m

class Trigonometria:
    '''Класс функций, способный вычислять:
    - синус
    - косинус
    - тангенс
    - арксинус
    - арккосинус
    - арктангенс'''


    def sin(x: int|float) -> float:
```

```
        '''Вычисляет синус числа'''
        if m.sin(x) < (m.e)**(-15) and m.sin(x) >= -(m.e)**(-15): # Если значени
            return 0.0
        else:
            return m.sin(x)

    def cos(x: int|float) -> float:
        '''Вычисляет косинус числа'''
        if m.cos(x) < (m.e)**(-15) and m.cos(x) >= -(m.e)**(-15):
            return 0.0
        else:
            return m.cos(x)

    def tg(x: int|float) -> float:
        '''Вычисляет тангенс числа'''
        if x % (m.pi/2) == 0 and x // (m.pi/2) % 2 != 0: # проверка на принадлеж
            return "ERROR"
        else:
            if m.tan(x) < (m.e)**(-15) and m.tan(x) >= -(m.e)**(-15):
                return 0.0
            else:
                return m.tan(x)

    def arcsin(x: int|float) -> float:
        '''Вычисляет арксинус числа'''
        if x >= -1 and x <= 1:
            if m.asin(x) < (m.e)**(-15) and m.asin(x) >= -(m.e)**(-15):
                return 0.0
            else:
                return m.asin(x)
        else:
            return "ERROR"

    def arccos(x: int|float) -> float:
        '''Вычисляет арккосинус числа'''
        if x >= -1 and x <= 1:
            if m.acos(x) < (m.e)**(-15) and m.acos(x) >= -(m.e)**(-15):
                return 0.0
            else:
                return m.acos(x)
        else:
            return "ERROR"

    def arctg(x: int|float) -> float:
        '''Вычисляет арктангенс числа'''
        if m.atan(x) < (m.e)**(-15) and m.atan(x) >= -(m.e)**(-15):
            return 0.0
        else:
            return m.atan(x)

print(Trigonometria.sin(2))
print(Trigonometria.cos(2))
print(Trigonometria.tg(m.pi))
print(Trigonometria.arcsin(2))
print(Trigonometria.arccos(0.5))
print(Trigonometria.arctg(2))
```

```
0.9092974268256817
-0.4161468365471424
0.0
ERROR
1.0471975511965979
1.1071487177940904
```

1.2.1 - Определите класс бинарного дерева и задайте его объекты с отдельным атрибутом для каждого из потомков.

In [ ]:
```
class Tree:
    def __init__(self, left, right):
        self.left = left
        self.right = right

t = Tree(Tree("a", "b"), Tree("c", "d"))
t.right.left
```

Out[ ]:
```
'c'
```

1.2.2 - Для обозначения отсутствующих потомков можно использовать None (в случае если у узла только один потомок). Само собой, можно комбинировать разные методы (например, использовать списки или множества потомков для каждого узла).

Распространенный способ реализации деревьев, особенно на языках, не имеющих встроенной поддержки списков, это так называемое представление «первый потомок, следующий брат». В нем каждый узел имеет два «указателя» или атрибута, указывающих на другие узлы, как в бинарном дереве. Однако, первый из этих атрибутов ссылается на первого потомка узла, а второй — на его следующего брата (т.е. узел, имеющий того же родителя, но находящийся правее, — прим. перев). Иными словами, каждый узел дерева имеет указатель на связанный список его потомков, а каждый из этих потомков ссылается на свой собственный аналогичный список. Таким образом, небольшая модификация бинарного дерева даст нам многопутевое дерево, показанное в листинге ниже.

In [ ]:
```
class Tree:
    def __init__(self, kids, next=None):
        self.kids = self.val = kids
        self.next = next

t = Tree(Tree("a", Tree("b", Tree("c", Tree("d")))))
t.kids.next.next.val
```

Out[ ]:
```
'c'
```

Задание - Представьте дерево показанное на рисунке с использованием списка из списков. Выведите на печать корень дерева, а также его левое и правое поддеревья.

In [ ]:
```
a = [["d", "e"], ["f"]]
b = a[0]
c = a[1]
```

```
print(a)
print(b)
print(c)
```

```
[['d', 'e'], ['f']]
['d', 'e']
['f']
```

Задание - Дан класс, описывающий бинарное дерево.

class Tree:

def **init**(self, data):

```
self.left = None

self.right = None

self.data = data
```

def PrintTree(self):

```
print(self.data)
```

Реализуйте в классе функцию для вставки нового элемента в дерево по следующим правилам:

• Левое поддерево узла содержит только узлы со значениями меньше, чем значение в узле.

• Правое поддерево узла содержит только узлы со значениями меньше, чем значение в узле.

• Каждое из левого и правого поддеревьев также должно быть бинарным деревом поиска.

• Не должно быть повторяющихся узлов.

Метод вставки сравнивает значение узла с родительским узлом и решает куда довавить элемент (в левое или правое поддерево). Перепишите, метод PrintTree для печати полной версии дерева.

In [ ]:
```
class Tree:
    def __init__(self, entry=None, left=None, right=None):
        self.entry = entry
        self.left = left
        self.right = right

    def put_in_tree(self, x):
        if self.entry is None:
            self.entry = x
        elif x < self.entry:
            if self.left is not None and self.right is not None:
                self.left.put_in_tree(x)
```

```
            elif self.left is not None:
                self.right = Tree(x)
            else:
                self.left = Tree(x)
        elif x == self.entry:
            pass
        else:
            if self.left is not None:
                self.left.put_in_tree(self.entry)
                self.entry = x
            else:
                self.entry = x



tree = Tree()
tree.put_in_tree(2)
tree.put_in_tree(12)
tree.put_in_tree(12)
tree.put_in_tree(5)
tree.put_in_tree(9)

tree
```

Out[ ]:
```
<__main__.Tree at 0x20806737150>
```

1.3.1 - Построим дерево решений для задачи классификации, для этого, построим границу решения для каждого класса. В качестве данных будем использовать уже знакомый нам и встроенный в библиотеку sklearn набор данных ирисов Фишера. Импортируем библиотеки, набор данных и посмотрим его характеристики.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
dataset = sns.load_dataset('iris')
dataset.shape
dataset.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Далее, разделим наши данные на атрибуты и метки, а затем выделим в общей совокупности полученных данных обучающие и тестовые наборы. Таким образом, мы можем обучить наш алгоритм на одном наборе данных, а затем протестировать его на совершенно на другом наборе, который алгоритм еще не видел. Это дает вам более точное представление о том, как на самом деле будет работать ваш обученный алгоритм.

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

X_train, X_test, y_train, y_test = train_test_split(
    # pandas-table -> iloc
    dataset.iloc[:, :-1],
    dataset.iloc[:, -1],
    test_size = 0.20
)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[ ]:
```
((120, 4), (30, 4), (120,), (30,))
```

In [ ]:
```
X_train.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width
13	4.3	3.0	1.1	0.1
4	5.0	3.6	1.4	0.2
52	6.9	3.1	4.9	1.5
27	5.2	3.5	1.5	0.2
46	5.1	3.8	1.6	0.2

In [ ]:
```
y_train.head()
```

Out[ ]:
```
13        setosa
4         setosa
52    versicolor
27        setosa
46        setosa
Name: species, dtype: object
```

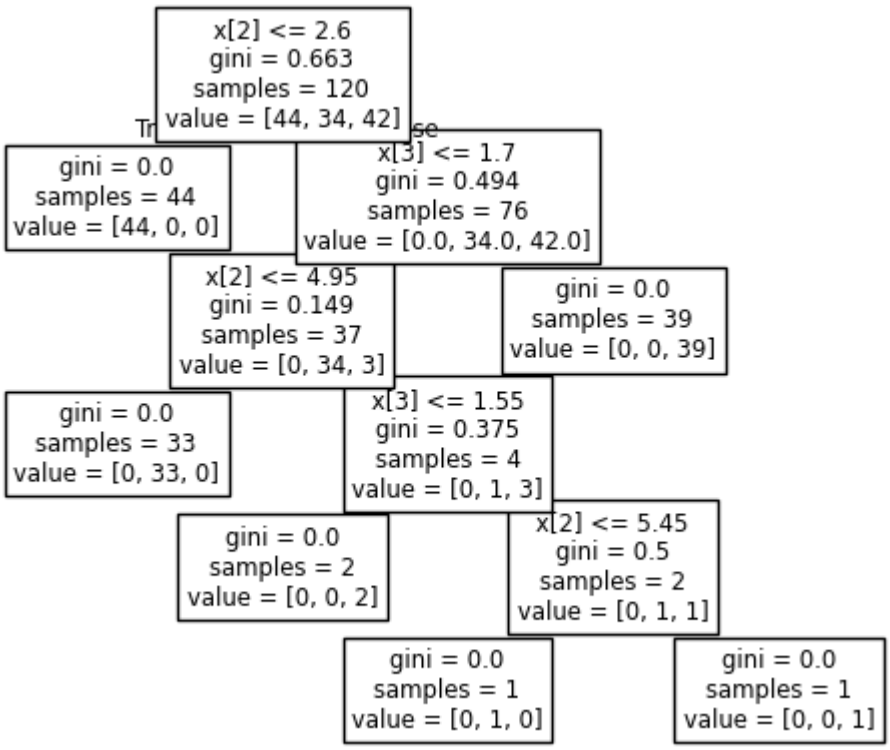
После того, как данные были разделены на обучающие и тестовые наборы, последний шаг состоит в том, чтобы обучить алгоритм дерева решений на этих данных и сделать прогнозы. Scikit-Learn содержит библиотеку tree , которая содержит встроенные классы/методы для различных алгоритмов дерева решений. Поскольку мы собираемся выполнить здесь задачу классификации, мы будем использовать класс DecisionTreeClassifier для этого примера. Метод fit этого класса вызывается для обучения алгоритма на обучающих данных, которые передаются в качестве параметра методу fit . Выполним следующий сценарий для обучения алгоритма.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:
```
# построим дерево решений
from sklearn import tree
tree.plot_tree(classifier)
```

Out[ ]:
```
[Text(0.3333333333333333, 0.9166666666666666, 'x[2] <= 2.6\ngini = 0.663\nsampl
es = 120\nvalue = [44, 34, 42]'),
 Text(0.16666666666666666, 0.75, 'gini = 0.0\nsamples = 44\nvalue = [44, 0,
0]'),
 Text(0.25, 0.8333333333333333, 'True  '),
 Text(0.5, 0.75, 'x[3] <= 1.7\ngini = 0.494\nsamples = 76\nvalue = [0.0, 34.0,
42.0]'),
 Text(0.41666666666666663, 0.8333333333333333, '  False'),
 Text(0.3333333333333333, 0.5833333333333334, 'x[2] <= 4.95\ngini = 0.149\nsamp
les = 37\nvalue = [0, 34, 3]'),
 Text(0.16666666666666666, 0.4166666666666667, 'gini = 0.0\nsamples = 33\nvalue
= [0, 33, 0]'),
 Text(0.5, 0.4166666666666667, 'x[3] <= 1.55\ngini = 0.375\nsamples = 4\nvalue
= [0, 1, 3]'),
 Text(0.3333333333333333, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.6666666666666666, 0.25, 'x[2] <= 5.45\ngini = 0.5\nsamples = 2\nvalue =
[0, 1, 1]'),
 Text(0.5, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.8333333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue
= [0, 0, 1]'),
 Text(0.6666666666666666, 0.5833333333333334, 'gini = 0.0\nsamples = 39\nvalue
= [0, 0, 39]')]
```



Теперь, когда наш классификатор обучен, давайте сделаем прогнозы по тестовым данным. Для составления прогнозов используется метод predict класса Decision Tree Classifier. Взгляните на следующий код для использования.

In [ ]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[ ]:
```
array(['versicolor', 'virginica', 'versicolor', 'setosa', 'versicolor',
       'virginica', 'virginica', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'setosa', 'versicolor', 'virginica', 'versicolor',
       'setosa', 'virginica', 'versicolor', 'versicolor', 'virginica',
       'setosa', 'virginica', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'versicolor', 'virginica'],
      dtype=object)
```

На данный момент мы обучили наш алгоритм и сделали некоторые прогнозы. Теперь посмотрим, насколько точен наш алгоритм. Для задач классификации обычно используются такие метрики, как матрица путаницы, точность. Библиотека Scikit-Learn metrics содержит методы classification_report и confusion_matrix, которые могут быть использованы для расчета этих метрик.

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

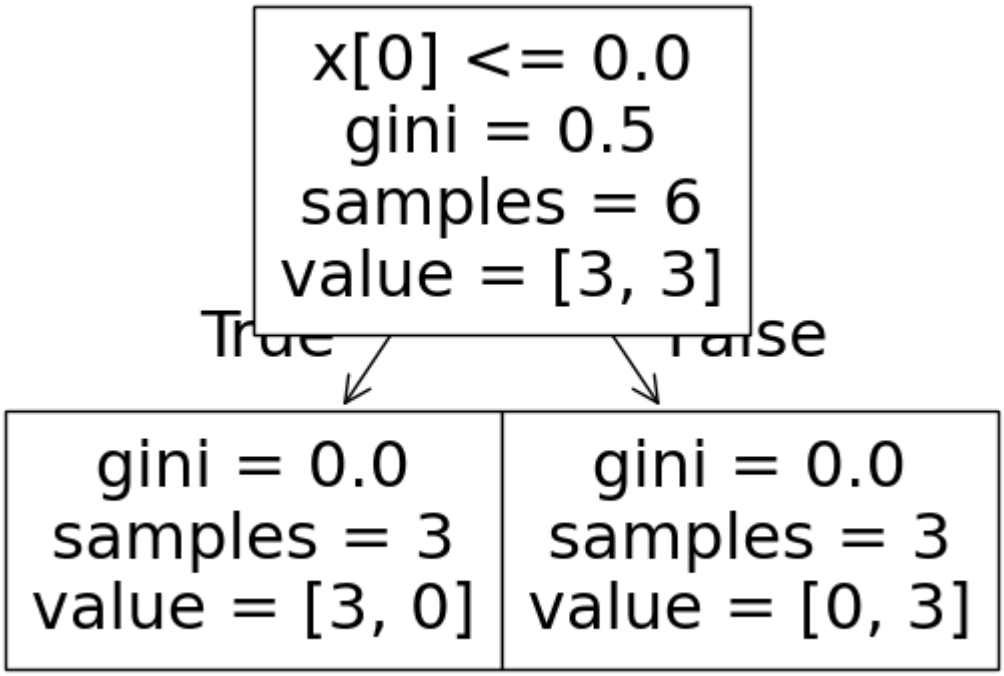
```
[[ 6  0  0]
 [ 0 14  2]
 [ 0  0  8]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         6
  versicolor       1.00      0.88      0.93        16
   virginica       0.80      1.00      0.89         8

    accuracy                           0.93        30
   macro avg       0.93      0.96      0.94        30
weighted avg       0.95      0.93      0.93        30
```

Из матрицы оценок алгоритма вы можете видеть, что из 30 тестовых экземпляров наш алгоритм неправильно классифицировал только 3. Это приблизительно 91 % точности.

Задание - Постройте классификатор на основе дерева принятия решений следующего датасета:

In [ ]:
```
# данные
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
target = [0, 0, 0, 1, 1, 1]

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X, target)

from sklearn import tree
tree.plot_tree(classifier)
```

Out[ ]:
```
[Text(0.5, 0.75, 'x[0] <= 0.0\ngini = 0.5\nsamples = 6\nvalue = [3, 3]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.375, 0.5, 'True  '),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.625, 0.5, '  False')]
```



1.4.1 - Постойте регрессию с использованием дерева решений, реализованного в Python.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
```

Out[ ]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [ ]:
```
# Исследуем набор данных
print(dataset.shape)
dataset.describe()
```

```
(30, 2)
```

Out[ ]:

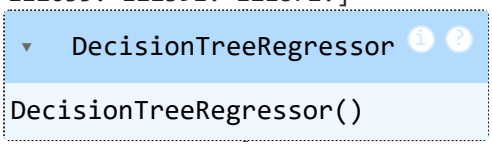
	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

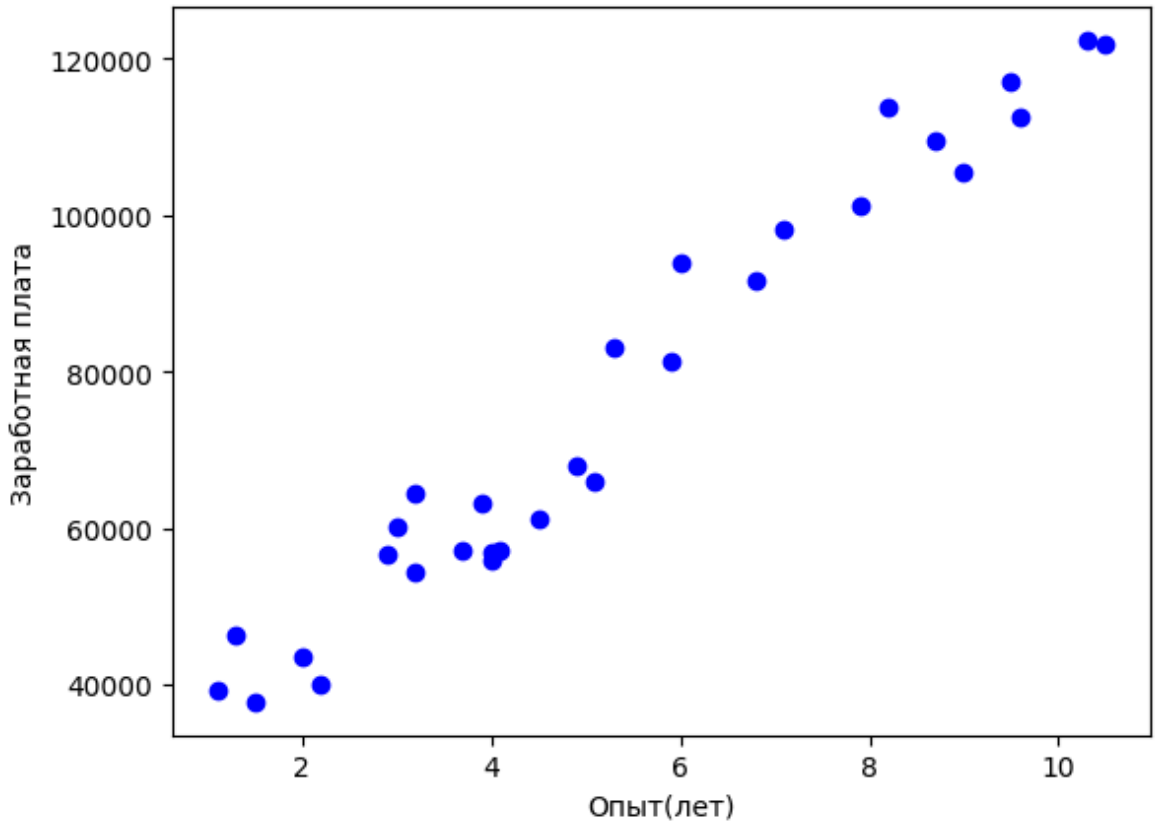
In [ ]:

```
# Нарисуем точечную диаграмму
plt.scatter(dataset['YearsExperience'], dataset['Salary'], color = 'b', label =
plt.xlabel("Опыт(лет)")
plt.ylabel("Заработная плата")

from sklearn.tree import DecisionTreeRegressor
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
print(X)
print(y)


# Теперь, когда у нас есть атрибуты и метки, необходимо разделить их на обучающи
# Приведенный фрагмент разделяет 80% данных на обучающий набор, а 20% - на набор
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Далее можно обучить алгоритм линейной регрессии
# Необходимо импортировать класс LinearRegression, создать его экземпляр и вызва
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
```

```
[[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]
 [ 8.2]
 [ 8.7]
 [ 9. ]
 [ 9.5]
 [ 9.6]
 [10.3]
 [10.5]]
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
  81363.  93940.  91738.  98273. 101302. 113812. 109431. 105582. 116969.
 112635. 122391. 121872.]
```

Out[ ]:

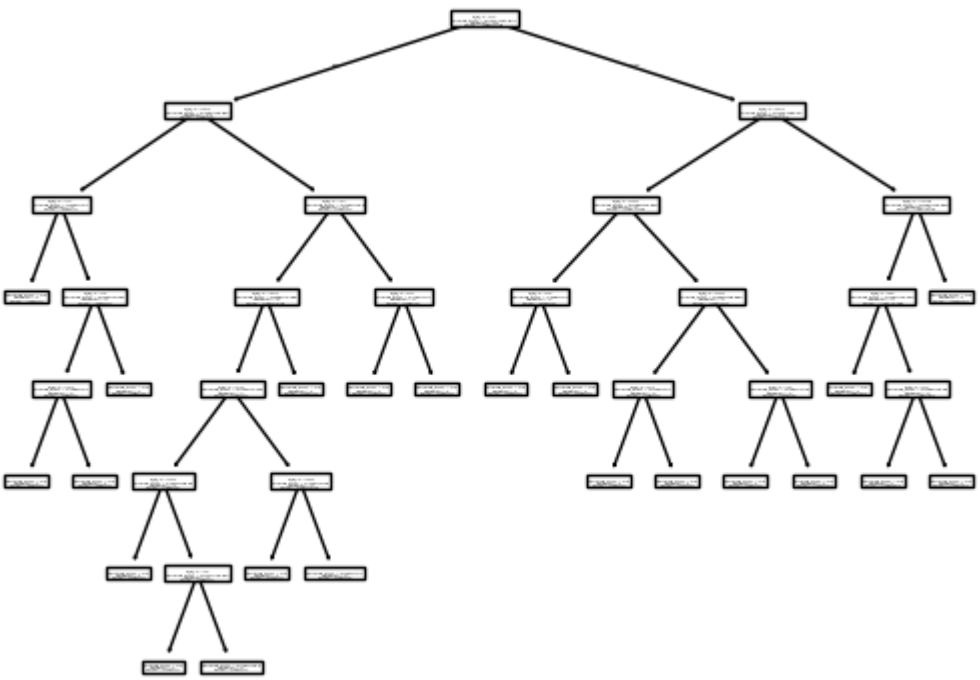




In [ ]:
```
from sklearn import tree
tree.plot_tree(regressor)
```

Out[ ]: [Text(0.4956896551724138, 0.9375, 'x[0] <= 5.2\nsquared_error = 614737637.832\nsamples = 24\nvalue = 73886.208'),
 Text(0.20689655172413793, 0.8125, 'x[0] <= 2.55\nsquared_error = 81200345.857\nsamples = 14\nvalue = 54976.0'),
 Text(0.35129310344827586, 0.875, 'True '),
 Text(0.06896551724137931, 0.6875, 'x[0] <= 1.2\nsquared_error = 7820714.0\nsamples = 4\nvalue = 42241.0'),
 Text(0.034482758620689655, 0.5625, 'squared_error = 0.0\nsamples = 1\nvalue = 39343.0'),
 Text(0.10344827586206896, 0.5625, 'x[0] <= 2.1\nsquared_error = 6694994.667\nsamples = 3\nvalue = 43207.0'),
 Text(0.06896551724137931, 0.4375, 'x[0] <= 1.65\nsquared_error = 1795600.0\nsamples = 2\nvalue = 44865.0'),
 Text(0.034482758620689655, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 46205.0'),
 Text(0.10344827586206896, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 43525.0'),
 Text(0.13793103448275862, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 39891.0'),
 Text(0.3448275862068966, 0.6875, 'x[0] <= 4.7\nsquared_error = 19731272.6\nsamples = 10\nvalue = 60070.0'),
 Text(0.27586206896551724, 0.5625, 'x[0] <= 4.25\nsquared_error = 9499922.484\nsamples = 8\nvalue = 58341.625'),
 Text(0.2413793103448276, 0.4375, 'x[0] <= 3.45\nsquared_error = 9604901.143\nsamples = 7\nvalue = 57946.0'),
 Text(0.1724137931034483, 0.3125, 'x[0] <= 2.95\nsquared_error = 14313358.25\nsamples = 4\nvalue = 58920.5'),
 Text(0.13793103448275862, 0.1875, 'squared_error = 0.0\nsamples = 1\nvalue = 56642.0'),
 Text(0.20689655172413793, 0.1875, 'x[0] <= 3.1\nsquared_error = 16777116.667\nsamples = 3\nvalue = 59680.0'),
 Text(0.1724137931034483, 0.0625, 'squared_error = 0.0\nsamples = 1\nvalue = 60150.0'),
 Text(0.2413793103448276, 0.0625, 'squared_error = 25000000.0\nsamples = 2\nvalue = 59445.0'),
 Text(0.3103448275862069, 0.3125, 'x[0] <= 3.85\nsquared_error = 372490.889\nsamples = 3\nvalue = 56646.667'),
 Text(0.27586206896551724, 0.1875, 'squared_error = 0.0\nsamples = 1\nvalue = 57189.0'),
 Text(0.3448275862068966, 0.1875, 'squared_error = 338142.25\nsamples = 2\nvalue = 56375.5'),
 Text(0.3103448275862069, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 61111.0'),
 Text(0.41379310344827586, 0.5625, 'x[0] <= 5.0\nsquared_error = 911070.25\nsamples = 2\nvalue = 66983.5'),
 Text(0.3793103448275862, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 67938.0'),
 Text(0.4482758620689655, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 66029.0'),
 Text(0.7844827586206896, 0.8125, 'x[0] <= 8.05\nsquared_error = 160167356.45\nsamples = 10\nvalue = 100360.5'),
 Text(0.6400862068965517, 0.875, ' False'),
 Text(0.6379310344827587, 0.6875, 'x[0] <= 5.95\nsquared_error = 53566814.556\nsamples = 6\nvalue = 91617.333'),
 Text(0.5517241379310345, 0.5625, 'x[0] <= 5.6\nsquared_error = 743906.25\nsamples = 2\nvalue = 82225.5'),
 Text(0.5172413793103449, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 83088.0'),
 Text(0.5862068965517241, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 81363.0'),

```
 Text(0.7241379310344828, 0.5625, 'x[0] <= 6.95\nsquared_error = 13823368.688\n
samples = 4\nvalue = 96313.25'),
 Text(0.6551724137931034, 0.4375, 'x[0] <= 6.4\nsquared_error = 1212201.0\nsamp
les = 2\nvalue = 92839.0'),
 Text(0.6206896551724138, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 93
940.0'),
 Text(0.6896551724137931, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 91
738.0'),
 Text(0.7931034482758621, 0.4375, 'x[0] <= 7.5\nsquared_error = 2293710.25\nsam
ples = 2\nvalue = 99787.5'),
 Text(0.7586206896551724, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 98
273.0'),
 Text(0.8275862068965517, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 10
1302.0'),
 Text(0.9310344827586207, 0.6875, 'x[0] <= 10.05\nsquared_error = 33407056.688
\nsamples = 4\nvalue = 113475.25'),
 Text(0.896551724137931, 0.5625, 'x[0] <= 8.6\nsquared_error = 13207004.222\nsa
mples = 3\nvalue = 110676.333'),
 Text(0.8620689655172413, 0.4375, 'squared_error = 0.0\nsamples = 1\nvalue = 11
3812.0'),
 Text(0.9310344827586207, 0.4375, 'x[0] <= 9.3\nsquared_error = 12436202.25\nsa
mples = 2\nvalue = 109108.5'),
 Text(0.896551724137931, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 105
582.0'),
 Text(0.9655172413793104, 0.3125, 'squared_error = 0.0\nsamples = 1\nvalue = 11
2635.0'),
 Text(0.9655172413793104, 0.5625, 'squared_error = 0.0\nsamples = 1\nvalue = 12
1872.0')]
```



In [ ]:
```
# Построим прогноз
y_pred = regressor.predict(X_test)
y_pred
```

Out[ ]:
```
array([ 46205. , 121872. ,  56375.5,  56375.5, 112635. , 105582. ])
```

In [ ]:
```
# теперь сравним некоторые из наших прогнозируемых значений с фактическими значе
df = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
```

```
df
```

Out[ ]:

	Actual	Predicted
0	37731.0	46205.0
1	122391.0	121872.0
2	57081.0	56375.5
3	63218.0	56375.5
4	116969.0	112635.0
5	109431.0	105582.0

In [ ]:
```
# Рассчитаем среднюю абсолютную и среднеквадратичную ошибку регрессии:
from sklearn import metrics
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

```
Mean Squared Error: 25498988.416666668
Mean Absolute Error: 4120.666666666667
```

In [ ]:
```
metrics.mean_absolute_error(y_test, y_pred) / np.average(y) * 100
```

Out[ ]:
```
5.421715809463662
```

Задание. Постройте модель регрессии для данных из предыдущей рабочей тетради.Для примера можно взять потребления газа (в миллионах галлонов) в 48 штатах США или набор данных о качестве красного вина: https://raw.githubusercontent.com/likarajo/petrol_consumption/master/data/petrol_consum https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv Постройте прогноз. Оцените точность модели.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('petrol_consumption.csv')
dataset.head()

# Исследуем набор данных
print(dataset.shape)
dataset.describe()

from sklearn.tree import DecisionTreeRegressor
print(X)
print(y)

# Теперь, когда у нас есть атрибуты и метки, необходимо разделить их на обучающи
# Приведенный фрагмент разделяет 80% данных на обучающий набор, а 20% - на набор
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Далее можно обучить алгоритм линейной регрессии
# Необходимо импортировать класс LinearRegression, создать его экземпляр и вызва
regressor = DecisionTreeRegressor()
```

```
regressor.fit(X_train, y_train)

# Построим прогноз
y_pred = regressor.predict(X_test)
print(y_pred)

# теперь сравним некоторые из наших прогнозируемых значений с фактическими значе
df = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
print(df)

# Рассчитаем среднюю абсолютную и среднеквадратичную ошибку регрессии:
from sklearn import metrics
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print(metrics.mean_absolute_error(y_test, y_pred) / np.average(y) * 100)
```

```
(48, 5)
[[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]
 [ 8.2]
 [ 8.7]
 [ 9. ]
 [ 9.5]
 [ 9.6]
 [10.3]
 [10.5]]
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
  81363.  93940.  91738.  98273. 101302. 113812. 109431. 105582. 116969.
 112635. 122391. 121872.]
[ 46205.  121872.   56375.5  56375.5 112635.  105582. ]
     Actual  Predicted
0   37731.0    46205.0
1  122391.0   121872.0
2   57081.0    56375.5
3   63218.0    56375.5
4  116969.0   112635.0
5  109431.0   105582.0
Mean Squared Error: 25498988.416666668
Mean Absolute Error: 4120.666666666667
5.421715809463662
```