02.Python intro

January 17, 2021

[1]:
```
myAge=20
print(myAge)
```

```
20
```

[2]:
```
myAge=21
print(myAge)
```

```
21
```

[3]:
```
print(myAge/3)
```

```
7.0
```

[4]:
```
myAge=myAge+1
print(myAge)
```

```
22
```

[5]:
```
restaurantBill =36.17
serviceCharge=0.125
print(restaurantBill*serviceCharge)
```

```
4.52125
```

[6]:
```
type(33)
```

[6]: int

[7]:
```
type(33.6)
```

[7]: float

[8]:
```
type("abc")
```

[8]: str

[9]:
```
type({"x":"y"})
```

[9]: dict

[10]:
```
type([1,2])
```

[10]: list

[11]:
```
type((1.2,3.4))
```

[11]: tuple

[12]:
```
coolPeople=["Nikhil","vikas","Aneesh"]
prime=[2,3,7]
primeAndPeople=["Nikhil",2,"Vikas"]
type(coolPeople)
type(prime)
type(primeAndPeople)
```

[12]: list

[13]:
```
prime[2]
```

[13]: 7

[14]:
```
import array as arr
prime=arr.array("i",[6,7,8])
type(prime)
```

[14]: array.array

[15]:
```
import pandas as pd
data =pd.read_csv("lsd_math_score_data.csv")
```

[16]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score
0                      5     1.17                78.93
1                     15     2.97                58.20
2                     30     3.26                67.47
3                     60     4.69                37.47
4                    120     5.83                45.65
5                    240     6.00                32.92
6                    480     6.41                29.97
```

[17]:
```
type(data)
```

[17]: pandas.core.frame.DataFrame

[18]:
```
onlyMathScores=data['Avg_Math_Test_Score']
```

[19]:
```
print(onlyMathScores)
```

```
0    78.93
1    58.20
2    67.47
3    37.47
4    45.65
5    32.92
6    29.97
Name: Avg_Math_Test_Score, dtype: float64
```

[20]:
```
data['Test_Subject']="Dayna Vendetta"
```

[21]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score    Test_Subject
0                      5     1.17                78.93  Dayna Vendetta
1                     15     2.97                58.20  Dayna Vendetta
2                     30     3.26                67.47  Dayna Vendetta
3                     60     4.69                37.47  Dayna Vendetta
4                    120     5.83                45.65  Dayna Vendetta
5                    240     6.00                32.92  Dayna Vendetta
6                    480     6.41                29.97  Dayna Vendetta
```

[22]:
```
data['High_Score']=100
```

[23]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score    Test_Subject  \
0                      5     1.17                78.93  Dayna Vendetta
1                     15     2.97                58.20  Dayna Vendetta
2                     30     3.26                67.47  Dayna Vendetta
3                     60     4.69                37.47  Dayna Vendetta
4                    120     5.83                45.65  Dayna Vendetta
5                    240     6.00                32.92  Dayna Vendetta
6                    480     6.41                29.97  Dayna Vendetta

   High_Score
0         100
1         100
2         100
3         100
4         100
5         100
6         100
```

[24]:
```
data["High_Score"]=data["High_Score"]+data["Avg_Math_Test_Score"]
```

[25]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score    Test_Subject  \
```

```
0                     5     1.17                78.93  Dayna Vendetta
1                    15     2.97                58.20  Dayna Vendetta
2                    30     3.26                67.47  Dayna Vendetta
3                    60     4.69                37.47  Dayna Vendetta
4                   120     5.83                45.65  Dayna Vendetta
5                   240     6.00                32.92  Dayna Vendetta
6                   480     6.41                29.97  Dayna Vendetta

   High_Score
0      178.93
1      158.20
2      167.47
3      137.47
4      145.65
5      132.92
6      129.97
```

[26]:
```
data['High_Score']=data['High_Score']*data['High_Score']
```

[27]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score    Test_Subject  \
0                      5     1.17                78.93  Dayna Vendetta
1                     15     2.97                58.20  Dayna Vendetta
2                     30     3.26                67.47  Dayna Vendetta
3                     60     4.69                37.47  Dayna Vendetta
4                    120     5.83                45.65  Dayna Vendetta
5                    240     6.00                32.92  Dayna Vendetta
6                    480     6.41                29.97  Dayna Vendetta

   High_Score
0  32015.9449
1  25027.2400
2  28046.2009
3  18898.0009
4  21213.9225
5  17667.7264
6  16892.2009
```

[28]:
```
type(onlyMathScores)
```

[28]:
```
pandas.core.series.Series
```

[29]:
```
# columnList=['LSD_ppm','Avg_Math_Test_Score']
cleanData=data[['LSD_ppm','Avg_Math_Test_Score']]
print(cleanData)
```

```
   LSD_ppm  Avg_Math_Test_Score
0     1.17                78.93
```

```
1     2.97                58.20
2     3.26                67.47
3     4.69                37.47
4     5.83                45.65
5     6.00                32.92
6     6.41                29.97
```

[30]:
```
y = data[['Avg_Math_Test_Score']]
```

[31]:
```
type(y)
```

[31]:
```
pandas.core.frame.DataFrame
```

[32]:
```
X=data[["LSD_ppm"]]
```

[33]:
```
print(X)
```

```
   LSD_ppm
0     1.17
1     2.97
2     3.26
3     4.69
4     5.83
5     6.00
6     6.41
```

[34]:
```
type(X)
```

[34]:
```
pandas.core.frame.DataFrame
```

[35]:
```
del data["Test_Subject"]
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score  High_Score
0                      5     1.17                78.93  32015.9449
1                     15     2.97                58.20  25027.2400
2                     30     3.26                67.47  28046.2009
3                     60     4.69                37.47  18898.0009
4                    120     5.83                45.65  21213.9225
5                    240     6.00                32.92  17667.7264
6                    480     6.41                29.97  16892.2009
```

[36]:
```
del data["High_Score"]
```

[37]:
```
print(data)
```

```
   Time_Delay_in_Minutes  LSD_ppm  Avg_Math_Test_Score
0                      5     1.17                78.93
1                     15     2.97                58.20
2                     30     3.26                67.47
```

```
3                    60      4.69                37.47
4                   120      5.83                45.65
5                   240      6.00                32.92
6                   480      6.41                29.97
```

[38]:
```
import life as lif
```

[39]:
```
type(lif)
```

[39]:
```
module
```

[40]:
```
lif.theAnswer
```

[40]:
```
42
```

[41]:
```
import math
print(math.pi)
print(math.e)
```

```
3.141592653589793
2.718281828459045
```

[42]:
```
math.inf
```

[42]:
```
inf
```

[43]:
```
-math.inf
```

[43]:
```
-inf
```

[44]:
```
type(lif)
```

[44]:
```
module
```

[45]:
```
from life import theAnswer
```

[46]:
```
theAnswer
```

[46]:
```
42
```

[47]:
```
theAnswer=theAnswer+1
print(theAnswer)
```

```
43
```

[48]:
```
type(theAnswer)
```

[48]:
```
int
```

[49]:
```
type(math.pi)
```

[49]:
```
float
```

[50]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

[51]:
```
def get_milk():
    print("Open door")
    print("Walk to the store")
    print("Buy milk on the ground floor")
    print("Return with milk galore")
```

[52]:
```
get_milk()
```

```
Open door
Walk to the store
Buy milk on the ground floor
Return with milk galore
```

[53]:
```
def fill_the_fridge(amount):
    print("Open door")
    print("Walk to the store")
    print("Buy " + amount +" cartons on the ground floor")
    print("Return with milk galore")
```

[54]:
```
fill_the_fridge('one thousand')
```

```
Open door
Walk to the store
Buy one thousand cartons on the ground floor
Return with milk galore
```

[55]:
```
#parameter creation placeholder variable
#argument using actual value given
```

[56]:
```
def milk_mission(amount,destination):
    print("Open door")
    print("Walk to the "+ destination)
    print("Buy " + amount +" cartons on the ground floor")
    print("Return with milk galore")
```

[57]:
```
milk_mission('twenty','department store')
```

```
Open door
Walk to the department store
Buy twenty cartons on the ground floor
Return with milk galore
```

[58]:
```
milk_mission(destination='department store',amount='twenty')
```

```
Open door
Walk to the department store
Buy twenty cartons on the ground floor
Return with milk galore
```

[59]:
```
milk_mission(destination='store',amount='twenty')
```

```
Open door
Walk to the store
Buy twenty cartons on the ground floor
Return with milk galore
```

[60]:
```
def get_milk(money):
    litres=money /1.15
    return litres
```

[61]:
```
get_milk(500)
```

[61]:
```
434.7826086956522
```

[62]:
```
amount=get_milk(20.5)
print(amount)
```

```
17.826086956521742
```

[63]:
```
def times(a,b):
    return a*b
```

[64]:
```
test=times(3.14,5.09)
print(test)
```

```
15.9826
```

[65]:
```
times('jk',4)
```

[65]:
```
'jkjkjkjk'
```

[66]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

```
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

objects

functions that are used with an objects are called as methods

[67]:
```
lif.quote_marvin()
```

```
 I've calculated your chance of survival, but I don't think you'll like it.
```

[76]:
```

```

[76]:
```
module
```

[68]:
```
myAge='Two keys'
type(myAge)
```

[68]:
```
str
```

[69]:
```
myAge=54.22
type(myAge)
```

[69]:
```
float
```

[70]:
```
import life as lif
```

[71]:
```
lif.square_root(63.14)
```

[71]:
```
7.946068210127573
```

[72]:
```
data
```

[72]:

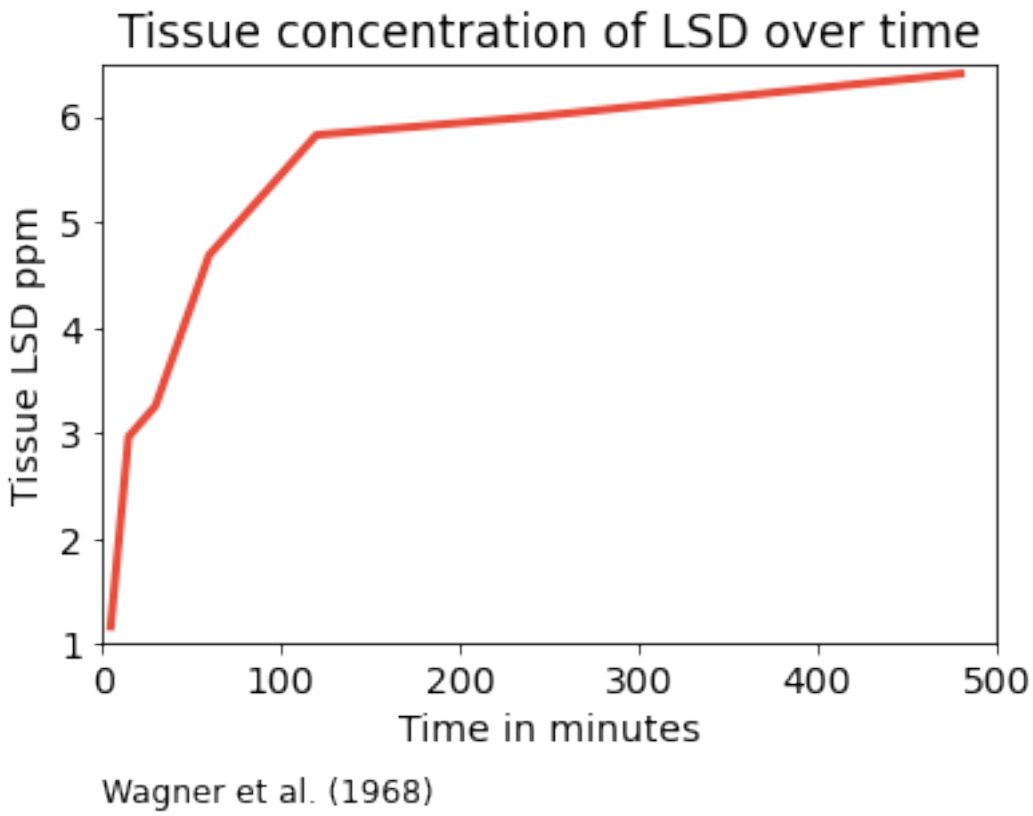
	Time_Delay_in_Minutes	LSD_ppm	Avg_Math_Test_Score
0	5	1.17	78.93
1	15	2.97	58.20
2	30	3.26	67.47
3	60	4.69	37.47
4	120	5.83	45.65
5	240	6.00	32.92

```
6                    480      6.41                 29.97
```

[73]:
```
time = data[['Time_Delay_in_Minutes']]
LSD = data[['LSD_ppm']]
score= data[['Avg_Math_Test_Score']]
```

[74]:
```
%matplotlib inline

plt.title('Tissue concentration of LSD over time',fontsize=17)
plt.xlabel('Time in minutes',fontsize=14)
plt.ylabel('Tissue LSD ppm',fontsize=14)
plt.text(x=0,y=-0.5,s="Wagner et al. (1968)",fontsize=12)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.ylim(1,6.5)
plt.xlim(0,500)
plt.style.use('classic')
plt.plot(time,LSD,c="#e74c3c",lw=3)
plt.show()
```



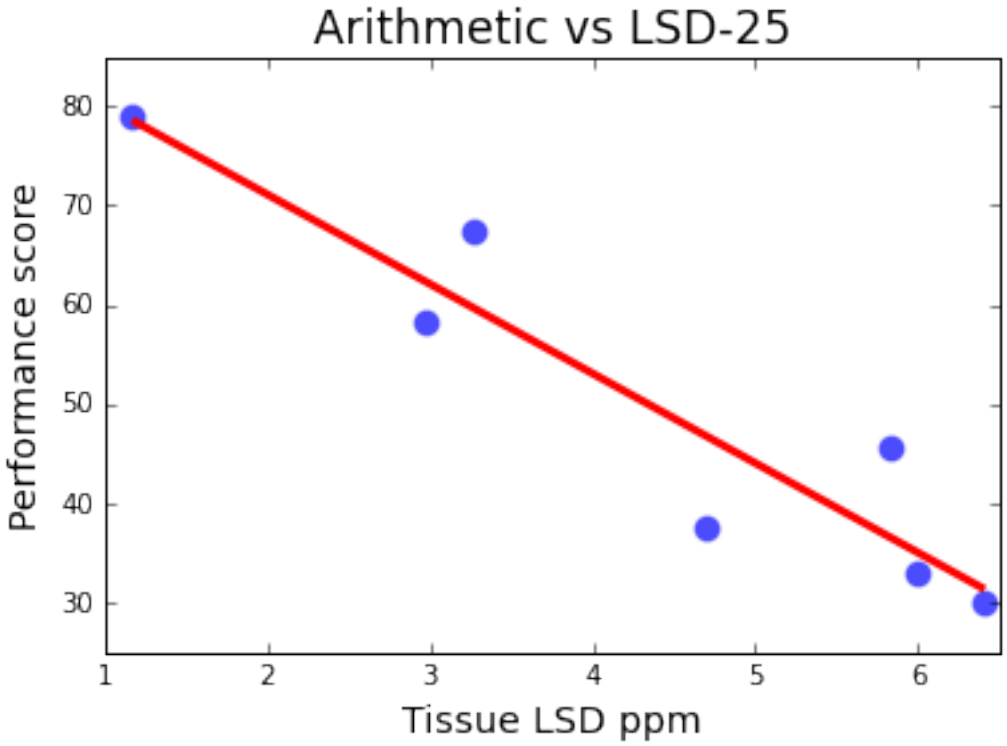
[75]:
```
regr=LinearRegression()
regr.fit(LSD,score)
print('Theta1 : ',regr.coef_[0][0])
print('Intercept : ',regr.intercept_[0])
print('R-square : ',regr.score(LSD,score))
predicted_score=regr.predict(LSD)
```

```
Theta1 :  -9.009466415296783
Intercept :  89.12387376799306
R-square :  0.8778349707775888
```

[76]:
```
%matplotlib inline

plt.title('Arithmetic vs LSD-25',fontsize=17)
plt.xlabel('Tissue LSD ppm',fontsize=14)
plt.ylabel('Performance score',fontsize=14)
plt.ylim(25,85)
plt.xlim(1,6.5)
plt.style.use('fivethirtyeight')

plt.scatter(LSD,score,c='blue',s=100,alpha=0.7)
plt.plot(LSD,predicted_score,c='r',lw='3')
plt.show()
```



[ ]: