Introduction to Python

Chapter 1 - Python Basics

In [ ]:
```
print(4+5)

# Subtraction
print(5-5)

# Multiplication
print(3*5)

# Division
print(10/2)
```

```
9
0
15
5.0
```

In [ ]:
```
# Create a variable savings
savings = 100

# Print out savings
print (savings)
```

```
100
```

In [ ]:
```
# Create the variables monthly_savings and num_months
monthly_savings = 10
num_months = 4

# Multiply monthly_savings and num_months, save the result as new_savings
new_savings = monthly_savings * num_months

# Add new_savings to your savings, save the sum as total_savings
total_savings = new_savings + savings

# Print total_savings
print(total_savings)
```

```
140
```

In [ ]:
```
# Create a variable half
half = 0.5
```

```
# Create a variable intro
intro = "Hello! How are you?"

# Create a variable is_good
is_good = True
```

In [ ]:

```
monthly_savings = 10
num_months = 12
intro = "Hello! How are you?"

# Calculate year_savings using monthly_savings and num_months
year_savings = monthly_savings*num_months

# Print the type of year_savings
print(type(year_savings))

# Assign sum of intro and intro to doubleintro
doubleintro = intro + intro

# Print out doubleintro
print(doubleintro)
```

```
<class 'int'>
Hello! How are you?Hello! How are you?
```

In [ ]:

```
# Definition of savings and total_savings
savings = 100
total_savings = 150

# Fix the printout
print("I started with $" + str(savings) + " and now have $" + str(total_savings) + ". Awesome!")

# Definition of pi_string
pi_string = "3.1415926"

# Convert pi_string into float: pi_float
pi_float = float(pi_string)
```

```
I started with $100 and now have $150. Awesome!
```

Chapter 2 - Python Lists

In [ ]:

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50
```

```
# Create list areas
areas = [hall, kit, liv, bed, bath]

# Print areas
print(areas)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [ ]:

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Adapt list areas
areas = ["hallway", hall, "kitchen", kit, "living room", liv, "bedroom", bed, "bathroom", bath]

# Print areas
print(areas)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75, 'bathroom', 9.5]
```

In [ ]:

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# house information as a list of lists
house = [["hallway", hall],
         ["kitchen", kit],
         ["living room", liv],
         ["bedroom", bed],
         ["bathroom", bath]]

# Print out house
print(house)

# Print out the type of house
print(type(house))
```

```
[['hallway', 11.25], ['kitchen', 18.0], ['living room', 20.0], ['bedroom', 10.75], ['bathroom', 9.5]]
<class 'list'>
```

In [ ]:

```
#Subsetting lists

# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]
```

```
# Print out second element from areas
print(areas[1])

# Print out last element from areas
print(areas[-1])

# Print out the area of the living room
print(areas[5])
```

```
11.25
9.5
20.0
```

In [ ]:

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Sum of kitchen and bedroom area: eat_sleep_area
eat_sleep_area = areas[3]+areas[-3]

# Print the variable eat_sleep_area
print(eat_sleep_area)
```

```
28.75
```

In [ ]:

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Use slicing to create downstairs
downstairs =areas[0:6]
# Alternative slicing to create downstairs
downstairs =areas[:6]


# Use slicing to create upstairs
upstairs =areas[6:10]
# Alternative slicing to create upstairs
upstairs =areas[6:]

# Print out downstairs and upstairs
print (downstairs)
print(upstairs)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]
['bedroom', 10.75, 'bathroom', 9.5]
```

In [ ]:

```
#manipulation Lists

# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Correct the bathroom area
```

```python
areas[9] = 10.50

# Change "living room" to "chill zone"
areas[4] = "chill zone"

print (areas)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5]
```

In [ ]:

```python
# Create the areas list and make some changes
areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,
         "bedroom", 10.75, "bathroom", 10.50]

# Add poolhouse data to areas, new list is areas_1
areas_1 = areas + ["poolhouse", 24.5]

# Add garage data to areas_1, new list is areas_2
areas_2 = areas_1 + ["garage", 15.45]

print(areas_1)
print(areas_2)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5, 'poolhouse', 24.5]
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5, 'poolhouse', 24.5, 'garage', 15.45]
```

In [ ]:

```python
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas_copy
areas_copy = list (areas) # areas_copy = areas[:]

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
print(areas_copy)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5]
[5.0, 18.0, 20.0, 10.75, 9.5]
```

Chapter 3 - Functions and Packages

In [ ]:

```python
# Create variables var1 and var2
var1 = [1, 2, 3, 4]
var2 = True

# Print out type of var1
print(type(var1))
```

```
# Print out length of var1
print(len(var1))

# Convert var2 to an integer: out2
out2 = int(var2)
print("true became an integer: ", out2)
```

```
<class 'list'>
4
true became an integer:  1
```

In [ ]:

```
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]

# Paste together first and second: full
full = first+second
print(full)

# Sort full in descending order: full_sorted
full_sorted = sorted(full, reverse = True)

# Print out full_sorted
print(full_sorted)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5]
[20.0, 18.0, 11.25, 10.75, 9.5]
```

In [ ]:

```
# string to experiment with: place
place = "poolhouse"

# Use upper() on place: place_up
place_up = place.upper()

# Print out place and place_up
print(place)
print(place_up)

# Print out the number of o's in place
print(place.count("o"))
```

```
poolhouse
POOLHOUSE
3
```

In [ ]:

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0
print(areas.index(20.0))
```

```
# Print out how often 9.50 appears in areas
print(areas.count(9.50))
```

```
2
1
```

In [ ]:
```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Use append twice to add poolhouse and garage size
areas.append(24.5)
areas.append(15.45)

# Print out areas
print(areas)

# Reverse the orders of the elements in areas
areas.reverse()

# Print out areas
print(areas)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5, 24.5, 15.45]
[15.45, 24.5, 9.5, 10.75, 20.0, 18.0, 11.25]
```

In [ ]:
```
# Import the math package
import math

# Definition of radius
r = 0.43

# Calculate C
C = 2*math.pi*r

# Calculate A
A = math.pi*pow(r,2)

# Build printout
print("Circumference: " + str(C))
print("Area: " + str(A))
```

```
Circumference: 2.701769682087222
Area: 0.5808804816487527
```

In [ ]:
```
# Import radians function of math package
from math import radians

# Definition of radius
r = 192500

# Travel angle of the Moon over 12 degrees
```

```
angle_degrees = 12

# Convert angle to radians using radians() function
angle_radians = radians(angle_degrees)

# Calculate the travel distance using the formula: distance = radius * angle
dist = r * angle_radians

# Print out dist
print(dist)
```

```
40317.10572106901
```

Chapter 4 - NumPy

The MLB also offers to let you analyze their weight data. Again, both are available as regular Python lists: height_in and weight_lb. height_in is in inches and weight_lb is in pounds.

It's now possible to calculate the BMI of each baseball player. Python code to convert height_in to a numpy array with the correct units is already available in the workspace. Follow the instructions step by step and finish the game! height_in and weight_lb are available as regular lists.

In [ ]:
```
import pandas as pd
mlb = pd.read_csv("C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course Da
height_in = mlb['Height'].tolist()
weight_lb = mlb['Weight'].tolist()
import numpy as np
```

In [ ]:
```
# Import the numpy package as np
import numpy as np

# Create list baseball
baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Create a numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out type of np_baseball
print(type(np_baseball))
```

```
<class 'numpy.ndarray'>
```

In [ ]:
```
# Import numpy
import numpy as np

# Create array from height_in with metric units: np_height_m
np_height_m = np.array(height_in) * 0.0254
```

```
# Create array from weight_lb with metric units: np_weight_kg
np_weight_kg = np.array(weight_lb) * 0.453592

# Calculate the BMI: bmi
bmi = np_weight_kg / np_height_m ** 2

# Print out bmi
print(bmi)
```

```
[23.11037639 27.60406069 28.48080465 ... 25.62295933 23.74810865
 25.72686361]
```

In [ ]:

```
# Import numpy
import numpy as np

# Calculate the BMI: bmi
np_height_m = np.array(height_in) * 0.0254
np_weight_kg = np.array(weight_lb) * 0.453592
bmi = np_weight_kg / np_height_m ** 2

# Create the light array
light = bmi < 21

# Print out light
print(light)

# Print out BMIs of all baseball players whose BMI is below 21
print(bmi[light])
```

```
[False False False ... False False False]
[20.54255679 20.54255679 20.69282047 20.69282047 20.34343189 20.34343189
 20.69282047 20.15883472 19.4984471  20.69282047 20.9205219 ]
```

In [ ]:

```
#Subsetting NumPy Arrays

# Import numpy
import numpy as np

# Store weight and height lists as numpy arrays
np_weight_lb = np.array(weight_lb)
np_height_in = np.array(height_in)

# Print out the weight at index 50
print(np_weight_lb[50])

# Print out sub-array of np_height_in: index 100 up to and including index 110
print(np_height_in[100:111])
```

```
200
[73 74 72 73 69 72 73 75 75 73 72]
```

Your First 2D NumPy Array

In [ ]:
```
# Import numpy
import numpy as np

# Create baseball, a list of lists
baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]

# Create a 2D numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out the type of np_baseball
print(type(np_baseball))

# Print out the shape of np_baseball
print(np_baseball.shape)
```

```
<class 'numpy.ndarray'>
(4, 2)
```

Subsetting 2D NumPy Arrays

If your 2D numpy array has a regular structure, i.e. each row and column has a fixed number of values, complicated ways of subsetting become very easy. Have a look at the code below where the elements "a" and "c" are extracted from a list of lists.

For regular Python lists, this is a real pain. For 2D numpy arrays, however, it's pretty intuitive! The indexes before the comma refer to the rows, while those after the comma refer to the columns. The : is for slicing; in this example, it tells Python to include all rows.

In [ ]:
```
#Example how to extract the first element from each nested list in a list

# regular list of lists
x = [["a", "b"], ["c", "d"]]
[x[0][0], x[1][0]]

# numpy
import numpy as np
np_x = np.array(x)
np_x[:, 0]
```

Out[ ]:
```
array(['a', 'c'], dtype='<U1')
```

In [ ]:
```
np_baseball
```

Out[ ]:
```
array([[180. ,  78.4],
       [215. , 102.7],
       [210. ,  98.5],
       [188. ,  75.2]])
```

In [ ]:
```
baseball = [[74, 180], [74, 215], [72, 210], [72, 210], [73, 188], [69, 176], [69, 209], [71, 200], [76, 231], [71, 180], [73, 188
```

In [ ]:
```
# Create np_baseball (2 cols)
np_baseball = np.array(baseball)

# Print out the 50th row of np_baseball
print(np_baseball[49,:])

# Select the entire second column of np_baseball: np_weight_lb
np_weight_lb=np_baseball[:,1]

# Print out height of 124th player
print(np_baseball[123,0])
```

```
[ 70 195]
75
```

2D Arithmetic

In [ ]:
```
# another example
np_mat = np.array([[1, 2],
                   [3, 4],
                   [5, 6]])
np_mat * 2
np_mat + np.array([10, 10])
np_mat + np_mat
```

Out[ ]:
```
array([[ 2,  4],
       [ 6,  8],
       [10, 12]])
```

In [ ]:
```
# Store weight and height lists as numpy arrays
np_weight_lb = np.array(weight_lb)
np_height_in = np.array(height_in)

# Print out the weight at index 50

print(np_weight_lb[50])
# Print out sub-array of np_height: index 100 up to and including index 110
print(np_height_in[100:111])
```

```
200
[73 74 72 73 69 72 73 75 75 73 72]
```

Average versus median

You now know how to use numpy functions to get a better feeling for your data. It basically comes down to importing numpy and then calling several simple functions on the numpy arrays:

```
import numpy as np

x = [1, 4, 8, 10, 12]

np.mean(x)

np.median(x)
```

The baseball data is available as a 2D numpy array with 3 columns (height, weight, age) and 200 rows. The name of this numpy array is np_baseball. After restructuring the data, however, you notice that some height values are abnormally high. Follow the instructions and discover which summary statistic is best suited if you're dealing with so-called outliers.

In [ ]:

```
avg = np.mean(np_baseball[:,0])
print("Average: " + str(avg))

# Print median height. Replace 'None'
med = np.median(np_baseball[:,0])
print("Median: " + str(med))

# Print out the standard deviation on height. Replace 'None'
stddev = np.std(np_baseball[:,0])
print("Standard Deviation: " + str(stddev))

# Print out correlation between first and second column. Replace 'None'
corr = np.corrcoef(np_baseball[:,0], np_baseball[:,1])
print("Correlation: " + str(corr))
```

```
Average: 73.83
Median: 74.0
Standard Deviation: 2.2893448844593074
Correlation: [[1.         0.55676088]
 [0.55676088 1.        ]]
```

Explore the baseball data

Because the mean and median are so far apart, you decide to complain to the MLB. They find the error and send the corrected data over to you. It's again available as a 2D Numpy array np_baseball, with three columns.

The Python script on the right already includes code to print out informative messages with the different summary statistics. Can you finish the job?

In [ ]:

```
# Create np_height_in from np_baseball
np_height_in=np_baseball[:,0]

# Print out the mean of np_height_in
print(np.mean(np_height_in))

# Print out the median of np_height_in
print(np.median(np_height_in))
```

```
73.83
74.0
```