CHAPTER – 1 - Class Resumes at 2:35 PM (PKT)

INTRO TO PYTHON

WHAT IS PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is regarded as one of the most essential tool for any Data Scientist.

WHAT YOU WILL LEARN

- Python
- Uses of Python in Data Science
- How to Store data
- How to Manipulate data
- Tools for data analysis

PYTHON

- Guido Van Rossum
- General Purpose: build anything
- Open Source! Free!
- Python Packages, also for Data Science
  - Many applications and fields
- Version 3.x - https://www.python.org/downloads/

PYTHON AS A CALCULATOR

Python is perfectly suited to do basic calculations. Apart from addition, subtraction, multiplication and division, there is also support for more advanced operations such as:

- Exponentiation: **. This operator raises the number to its left to the power of the number to its right. For example 4**2 will give 16.
- Modulo: %. This operator returns the remainder of the division of the number to the left by the number on its right. For example 18 % 7 equals 4.

TASKS

Suppose you have $100, which you can invest with a 10% return each year. After one year, it's 100×1.1=110 dollars, and after two years it's 100×1.1×1.1=121.

Add code to calculate how much money you end up with after 7 years, and print the result.

ANSWER

```
print (100 * 1.1 ** 7)
```

VARIABLE ASSIGNMENT

In Python, a variable allows you to refer to a value with a name. To create a variable use **=**, like this example:

x = 5

You can now use the name of this variable, x, instead of the actual value, 5.

Remember, = in Python means assignment, it doesn't test equality!

TASK

Create a variable savings with the value 100.

Check out this variable by typing print(savings) in the script.

ANSWER

```
savings = 100
print(savings)
```

CALCULATIONS WITH VARIABLES

Remember how you calculated the money you ended up with after 7 years of investing $100? You did something like this: 100 * 1.1 ** 7

Instead of calculating with the actual values, you can use variables instead. The savings variable you've created in the previous exercise represents the $100 you started with.

CALCULATIONS WITH VARIABLES EXAMPLE

- Specific, case-sensitive name
- Call up value through variable name
- 1.79 m - 68.7 kg

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79
```

CALCULATIONS WITH VARIABLES EXAMPLE

```
In [1]: height = 1.79

In [2]: weight = 68.7

In [3]: height
Out[3]: 1.79

In [4]: 68.7 / 1.79 ** 2
Out[4]: 21.4413

In [5]: weight / height ** 2
Out[5]: 21.4413

In [6]: bmi = weight / height ** 2

In [7]: bmi
Out[7]: 21.4413
```

Calculate BMI

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

TASKS

Create a variable growth_multiplier, equal to 1.1.

Create a variable, result, equal to the amount of money you saved after 7 years.

Print out the value of result.

ANSWER

```
#create variable growth_multiplier
growth_multiplier =1.1
# create variable  result and save
result = savings * growth_multiplier ** 7
#Print out
print(result)
```

OTHER VARIABLE TYPES

In the previous exercise, you worked with two Python data types:

int, or integer: a number without a fractional part. **savings**, with the value **100**, is an example of an integer.

float, or floating point: a number that has both an integer and fractional part, separated by a point. **growth_multiplier**, with the value **1.1**, is an example of a float.

OTHER VARIABLE TYPES

Next to numerical data types, there are two other very common data types:

str, or string: a type to represent text. You can use single or double quotes to build a string.

bool, or boolean: a type to represent logical values. Can only be **True** or **False** (the capitalization is important!).

TASK

Create a new string, desc, with the value "compound interest".

ANSWER

desc = 'compound interest'

TASKS

Create a new boolean, profitable, with the value True.

ANSWER

profitable = True

HOW TO FIND OUT VARIABLE TYPES

To find out the type of a value or a variable that refers to that value, you can use the type() function. Suppose you've defined a variable a, but you forgot the type of this variable.

To determine the type of a, simply execute: type(growth_multiplier)

OPERATIONS WITH OTHER TYPES

When you sum two strings, for example, you'll get different behavior than when you sum two integers or two booleans.

TASKS

Calculate the product of savings and growth_multiplier. Store the result in year1. What do you think the resulting type will be? Find out by printing out the type of year1.

ANSWER

```
#Assign product of savings and growth_multiplier to year1
year1=savings * growth_multiplier
#print year1
print(year1)
#print type of year1
print(type(year1))
```

TASKS

Calculate the sum of desc and desc and store the result in a new variable doubledesc. Print out doubledesc. Did you expect this?

ANSWER

```
# Assign sum of desc and desc to doubledesc
 doubledesc= desc+desc
# Print out doubledesc
 print(doubledesc)
```

TYPE CONVERSION

Using the **+** operator to paste together two strings can be very useful in building custom messages.

Example

Suppose, that you've calculated the return of your investment and want to summarize the results in a string. Assuming the floats **savings** and **result** are defined, you can try something like this:

print("I started with $" + savings + " and now have $" + result + ". Awesome!")

This will not work, though, as you cannot simply sum strings and floats.

TYPE CONVERSION

To fix the error, you'll need to explicitly convert the types of your variables. More specifically, you'll need str(), to convert a value into a string. str(savings), for example, will convert the float savings to a string.

Similar functions such as int(), float() and bool() will help you convert Python values into any type.

TASK

Create a variable pi_string with “3.1415926” value. print the type of pi_string.

ANSWER

```
pi_string = "3.1415926"
print(type(pi_string))
```

TASK

Convert the variable pi_string to a float and store this float as a new variable, pi_float. print the type of pi_float.

ANSWER

```
pi_float = float(pi_string)
print(type(pi_float))
```

List

LISTS IN PYTHON

In Python, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.). Also, a list can even have another list as an item. This is called nested list.

CREATING A LIST

As opposed to int, bool etc., a list is a compound data type; you can group values together:

```
a = "is"
b = "nice"
my_list = ["my", "list", a, b]
```

CREATING A LIST - EXAMPLE

After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are shown in the script.

```
hall=11.25
kit=18.0
liv=20.0
bed=10.75
bath=9.50
```

TASKS

Create a list, areas, that contains the area of the hallway (hall=11.25), kitchen (kit=18.0), living room (liv=20.0), bedroom (bed=10.75) and bathroom (bath=9.50), in this order. Use the predefined variables.

Print areas with the print() function.

ANSWER

```
#area variable (in square meters)
hall=11.25
kit=18.0
liv=20.0
bed=10.75
bath=9.50
#create list area
areas=[hall,kit,liv,bed,bath]
#print areas
print(areas)
```

CREATE LIST WITH DIFFERENT TYPES

A list can contain any Python type. Although it's not really common, a list can also contain a mix of Python types including strings, floats, booleans, etc.

CREATE LIST WITH DIFFERENT TYPES - EXAMPLE

The printout of the previous exercise wasn't really satisfying. It's just a list of numbers representing the areas, but you can't tell which area corresponds to which part of your house.

The code below is the start of a solution. For some of the areas, the name of the corresponding room is already placed in front. Pay attention here! **"bathroom"** is a string, while **bath** is a variable that represents the float **9.50** you specified earlier.

TASKS

Finish the line of code that creates the **areas** list. Build the list so that the list first contains the name of each room as a string and then its area. In other words, add the strings **"hallway", "kitchen" and "bedroom"** at the appropriate locations. Print **areas** again.

ANSWER

```
#area variable (in square meters)
hall=11.25
kit=18.0
liv=20.0
bed=10.75
bath=9.50
#create list area
areas=["hallway",hall,"kitchen",kit,"living room",liv,"bedroom",bed,"bathroom",bath]
#print areas
print(areas)
```

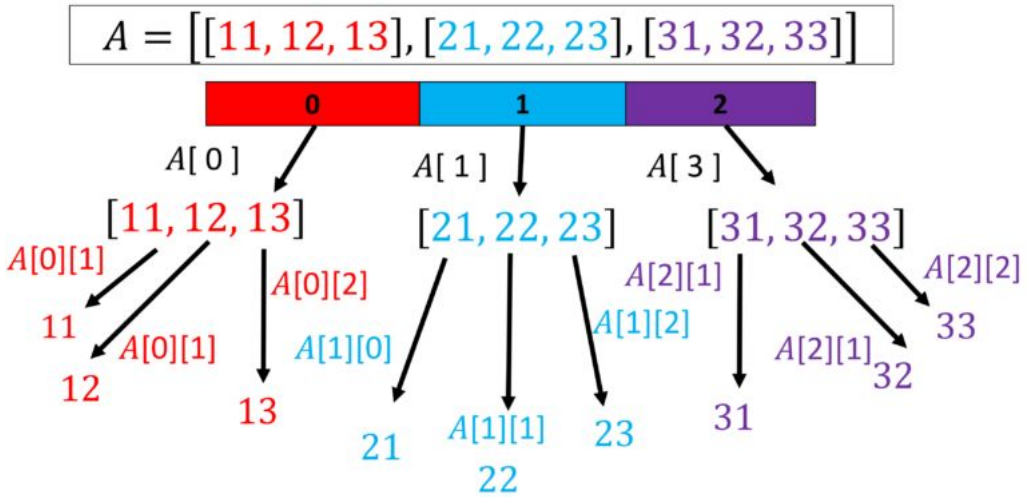
SELECT A VALID LIST

A list can contain any Python type. But a list itself is also a Python type. That means that a list can also contain a list! Python is getting funkier by the minute, but fear not, just remember the list syntax:

```
my_list = [el1, el2, el3]
```

LIST OF LISTS

- In Python any table can be represented as a list of lists (a list, where each element is in turn a list).
- As a data scientist, you'll often be dealing with a lot of data, and it will make sense to group some of this data.



LIST OF LISTS - EXAMPLE

Using a list of lists

```
#creating a list of list using area variables (in square meters) for 3 houses
House1 [11.25,18.0, 20.0, 10.75, 9.50]
House2 [3.25,8.0, 10.0, 7.20, 3.50]
House3 [4.25,6.0, 13.50, 7.80, 7.50]

#creating a list of list for 3 houses

house_list = [[11.25,18.0, 20.0, 10.75, 9.50], [3.25,8.0, 10.0, 7.20, 3.50], [4.25,6.0, 13.50, 7.80, 7.50]]

#You have Created a List of List Above.

#print out house_list

print (house_list)

#print out the type of house

print(type(house_list))
```

TASKS

Can you structure the house data as a list of list. Create a List Called house and also print the type of the list.

hall=11.25
kit=18.0
liv=20.0
bed=10.75
bath=9.50

Your Output Should Look Like This:

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

ANSWER

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

SUBSETS

Subsetting Python lists is a piece of cake. Take the code sample below, which creates a list x and then selects "b" from it. Remember that this is the second element, so it has index 1. You can also use negative indexing.

So if, x = ["a", "b", "c", "d"]

```
x[1]
```

Out: b

```
x[-3]
#same result!
```

Out: b

TASKS

Print out the second element from the areas list (it has the value 11.25).

Use the following areas list:

areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
#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 form areas
print(areas[1])
```

TASKS

Print out the last element of areas, being 9.50. Using a negative index makes sense here!

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
#create the areas list
areas=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",9.50]
#Print out last element form areas
print(areas[-1])
```

TASKS

Select the number representing the area of the living room (20.0) and print it out.

Use the following areas list:

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

ANSWER

```
#create the areas list
areas=["hallway",11.25,"kitchen",18.0,"living
room",20.0,"bedroom",10.75,"bathroom",9.50]
#Print out the area of living room
print(areas[5])
```

SUBSETS & CALCULATE

After you've extracted values from a list, you can use them to perform additional calculations.

Take this example, where the second and fourth element of a list x are extracted. The strings result are pasted together using the + operator:

```
x = ["a", "b", "c", "d"]
print(x[1] + x[3])
```

Out:

```
bd
```

TASKS

Using a combination of list subsetting and variable assignment, create a new variable, **eat_sleep_area**, that contains the sum of the area of the **kitchen** and the area of the **bedroom**.

Print the new variable **eat_sleep_area**.

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
#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 badroom area: eat_sleep_area
eat_sleep_area=areas[3]+areas[7]
#Print the variable eat_sleep_area
print(eat_sleep_area)
```

SLICING & DICING

Selecting single values from a list is just one part of the story. It's also possible to slice your list, which means selecting multiple elements from your list. Use the following syntax:

```
my_list[start:end]
```

SLICING & DICING

The start index will be included, while the end index is not.

The code sample below shows an example. A list with "b" and "c", corresponding to indexes 1 and 2, are selected from a list x:

```
x = ["a", "b", "c", "d"]
x[1:3]
```

Out:

```
['b', 'c']
```

The elements with index 1 and 2 are included, while the element with index 3 is not.

TASKS

Use slicing to create a list, downstairs, that contains the first 6 elements of areas.

Print downstairs using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
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]

#print out downstairs
print(downstairs)
```

TASKS

Do a similar thing to create a new variable, upstairs, that contains the last 4 elements of areas.

Print upstairs using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

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

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

#print out upstairs
print(upstairs)
```

SLICING & DICING

You can specify both where to begin and end the slice of your list:

```
my_list[begin:end]
```

However, it's also possible not to specify these indexes.

- If you don't specify the begin index, Python figures out that you want to start your slice at the beginning of your list.
- If you don't specify the end index, the slice will go all the way to the last element of your list.

To experiment with this, try the following commands in Juypter:

```
x = ["a", "b", "c", "d"]
x[:2]
Out: ['a', 'b']
x[2:]
Out: ['c', 'd']
x[:]
Out: ['a', 'b', 'c', 'd']
```

TASKS

Create downstairs again, as the first 6 elements of areas. This time, simplify the slicing by omitting the begin index.

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
areas=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",9.50]
#Use slicing to create downstairs
downstairs = areas[:6]


#print out downstairs
print(downstairs)
```

TASKS

Create upstairs again, as the last 4 elements of areas. This time, simplify the slicing by omitting the end index.

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

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


#Use slicing to create upstairs
upstairs = areas[-4:]
#print out  upstairs
print(upstairs)
```

SUBSETTING LISTS OF LISTS

You saw before that a Python list can contain practically anything; even other lists! To subset lists of lists, you can use the same technique as before: square brackets. Try out the commands in the following code sample:

```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
```

So, let's try:

```
x[2][0]
```

Out: 'g'

```
x[2][:2]
```

Out: ['g', 'h']

UPDATING VALUES IN A LISTS OF LISTS

You saw before that a Python list can contain practically anything; even other lists! To subset lists of lists, you can use the same technique as before: square brackets. Try out the commands in the following code sample:

```
x = [["a", "b", "c"],
     ["d", "e", "f"],
     ["g", "h", "i"]]
```

So, let's try to update the value of d to dd:

```
x[1][0] = 'dd'
#print x
print(x)
```

Out: [['a', 'b', 'c'], ['dd', 'e', 'f'], ['g', 'h', 'i']]

TASKS

Update the area of the bathroom area to be 10.50 square meters instead of 9.50. Print out areas list using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
areas=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",9.50]
#Correct the bathroom ares
areas[-1]=10.50
#print out areas list using print().
print(areas)
```

TASKS

Make the areas list more trendy! Change "living room" to "chill zone". Print out areas list using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
areas=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",10.5]

#change “living room ” to “ chill zone”
areas[4]="chill zone"
#print out areas list using print().
print(areas)
```

EXTEND A LIST

If you can change elements in a list, you sure want to be able to add elements to it, right? You can use the + operator:

Let's try adding values to a list:

```
x = ["a", "b", "c", "d"]
y = x + ["e", "f"]
```

Out: ["a", "b", "c", "d", "e", "f"]

TASKS

You just won the lottery, awesome! You decide to build a poolhouse. Can you add the information to the areas list?

Hint: Use the + operator to paste the list ["poolhouse",24.5] to the end of the areas list. Store the resulting list as areas_1.

print area_1 list using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
areas=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",9.50]
#Add poolhouse data to areas, new list is area_1
area_1=areas+["poolhouse",24.5]
#print area_1
print(area_1)
```

TASKS

You decide to build a garage [15.45]. Can you add the information to the areas list?

print areas list using print().

Use the following areas list:
areas=["hallway",11.25,"kitchen",18.0,"living Room",20.0,"bedroom",10.75,"bathroom",9.50]

ANSWER

```
areas_1=["hallway",11.25,"kitchen",18.0,"living room",20.0,"bedroom",10.75,"bathroom",9.50,"poolhouse",24.5]
#Add garage data to areas_1, new list is area_2
area_2=area_1+["garage",15.45]
#print areas_2
print(area_2)
```

DELETING LIST ELEMENTS

Finally, you can also remove elements from your list. You can do this with the del statement:

Lets try to delete b from the list x below:

```
x = ["a", "b", "c", "d"]
del(x[1])
#print out x
Print(x)
Out:
["a", "c", "d"]
```

Pay attention here: as soon as you remove an element from a list, the indexes of the elements that come after the deleted element all change!

DELETING LIST ELEMENTS

The updated and extended version of areas that you've built in the previous exercises is coded below.

```
areas = ["hallway", 11.25, "kitchen", 18.0,
         "chill zone", 20.0, "bedroom", 10.75,
          "bathroom", 10.50, "poolhouse", 24.5,
          "garage", 15.45]
```

TASKS

Delete garage and its value form areas list.

The updated and extended version of areas that you've built in the previous exercises is coded below.

```
areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0, "bedroom", 10.75, "bathroom", 10.50, "poolhouse", 24.5, "garage", 15.45]
```

ANSWER

```
areas=["hallway",11.25,"kitchen",18.0,"chill zone",20.0,"bedroom",10.75,"bathroom", 10.50, "poolhouse", 24.5,"garage", 15.45]
#delete garage and its float value
del(areas[13])
del(areas[12])
#Print updated areas list
print (areas)
```

Q&A

python

THANK YOU