Pair Programming Exercises on Lists

Created by HD Sheets, July 2024 for DSE5002

Updated 11/13/2024

Resources and Sources

https://developers.google.com/edu/python/lists

This an introduction to lists, lists are

-iterable- meaning you can use them in a for loop or a comprehension they have an order to them and can be sorted

-mutable- they can be altered after being created

Think Python Book

https://allendowney.github.io/ThinkPython/chap09.html

Lists in python work much as they do in R

They are created by placing a set of variables inside square brackets

In [22]:
```
a=[1,2,3,4,5,6]
clients=["John", "Angel","Rafael"]
```

In [23]:
```
# we can index by numeric location
# remember indexing starts a zero

a[1]
```

Out[23]: 2

In [24]:
```
clients[0]
```

Out[24]: 'John'

Question/Action

How would you index 1 and Rafael in the two lists?

Add a code cell and show this

In [25]:
```
print(a[0],clients[-1])
```

```
1 Rafael
```

Keeping track of data types

As you work with Python packages, and learn from example code, one thing you need to always be aware of is what the data structure used in an example is.

If a neural net expects a numpy matrix that of k rows and 784 columns, that's the format you need to get the code to run on your data.

To format your data correctly, you need to be constantly aware of the data format in use, and that is often not obvious in Python.

In [26]:
```
type(a)
```

Out[26]:
```
list
```

In [27]:
```
type(clients)
```

Out[27]:
```
list
```

Use dir() to see what built in functions and variables are avaiable for lists

In [28]:
```
dir(clients)
```

Out[28]:
```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [29]:
```
# append adds on item to a list

clients.append("Fan")
clients
```

Out[29]:
```
['John', 'Angel', 'Rafael', 'Fan']
```

In [30]:
```
# remove an entry
clients.remove("John")
clients
```

Out[30]:
```
['Angel', 'Rafael', 'Fan']
```

Aliases versus copies

or "shallow copies" verses "deep copies"

In [31]:
```
#  setting an alias

clients2=clients

# clients2 is just another name, an alias, for clients.  It is not a copy of the da

clients2.append("Fran")

#notice the append was to clients2, not clients, but look what happened to clients
clients
```

Out[31]:
```
['Angel', 'Rafael', 'Fan', 'Fran']
```

In [32]:
```
# tp create a second copy of a list, we need the copy function

clients3=clients.copy()

clients3.remove("Angel")

clients
```

Out[32]:
```
['Angel', 'Rafael', 'Fan', 'Fran']
```

In [33]:
```
# just to check that the remove worked

clients3
```

Out[33]:
```
['Rafael', 'Fan', 'Fran']
```

In [34]:
```
#add items at a specific location

clients.insert(0,"Kim")
clients
```

Out[34]:
```
['Kim', 'Angel', 'Rafael', 'Fan', 'Fran']
```

Pushing and Popping

Stack-like operations

In [ ]:
```
# pop removes the last item in a list and allow you to put it in a new variable
#  the value is removed form the list
#  this is an implementation of an older programming structure called a "stack"
# you pushed objects on a stack - appending them to end
# and popped them back off, removing them from the end

d=clients.pop()
print(d)
print(clients)
```

```
Fran
['Kim', 'Angel', 'Rafael', 'Fan']
```

In [ ]:
```
# we can sort lists
clients.sort()

#note: this is a sort in place, it doesn't create a sorted copy,  it sorts the vari

print(clients)
```

```
['Angel', 'Fan', 'Kim', 'Rafael']
```

In [ ]:
```
clients.reverse()
clients
```

Out[ ]:
```
['Rafael', 'Kim', 'Fan', 'Angel']
```

In [ ]:
```
clients3.sort(reverse=True)
```

In [ ]:
```
clients3
```

Out[ ]:
```
['Rafael', 'Fran', 'Fan']
```

Lists are iterable

We can use them in a for loop, as shown below

In [ ]:
```
for name in clients:
    print(name)
```

```
Rafael
Kim
Fan
Angel
```

Notice the syntax

we have an iterator (name) that takes on each value in clients, one at a time

the for statement ends with a :

the body of commands in the loop is indented- python uses indentation to denote code blocks, be careful with this

Question/Action

Here is a list of numbers

Add code to print the number and it's square

just go in alter my code

In [36]:
```
x=[1,2,3,4,5,6,7,8,9]

for val in x:
    print(val, val**2)
```

```
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
```

LIst comprehensions are quick ways to run a loop like operation

In [ ]:
```
[print(var,var**2) for var in x]
```

```
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
```

Out[ ]:
```
[None, None, None, None, None, None, None, None, None]
```

In [ ]:
```
# we can create a list using a comprehension

x_cubed=[var**3 for var in x]

x_cubed
```

Out[ ]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```