Chapter 1: CMD guide

```
C:\Users\fayaz>dir
```

shows directory of files

```
C:\Users\fayaz\OneDrive>cd Desktop
```

changes directory to 'Desktop'

```
C:\Users\fayaz\OneDrive\Desktop\Code>axb.py
```

opens file 'axb.py' in folder 'Code'
save files with .py extension

```
C:\Users\fayaz\Desktop>cd..

C:\Users\fayaz>
```

goes back

Consider using the `--user` option or check the permissions.

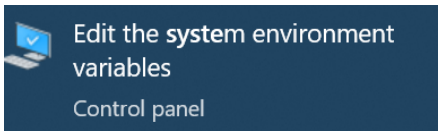
to upgrade pip

```
pip install -U  --upgrade pip
```

To add the path of your pip installation to your PATH variable, you can use the Control Panel or the `setx` command. For example:

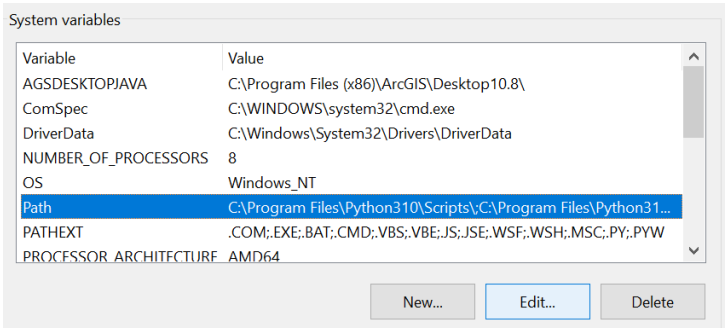
```
setx PATH "%PATH%;C:\Python34\Scripts"
```

if it's not working, try this



or this:

add path manually



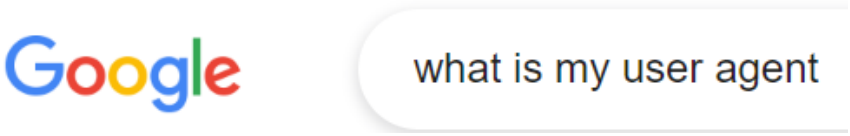
For Windows, when you install a package, you type:

```
python -m pip install [packagename]
```

or after adding the path, pip should work just fine

```
C:\Users\fayaz\OneDrive\Desktop\Code>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

type python for version



Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36

CTRL + C to kill cmd processs

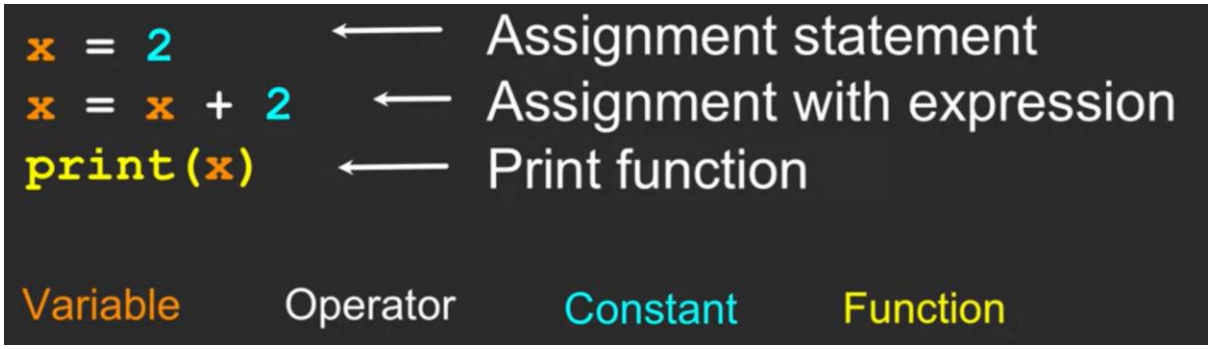
Chapter 2: Python guide

2.1: Variables

```
>>> x = 1
>>> print (x)
1
```

1 is being put in to the variable named ‘x’.

‘print’ function prints out what is stored in variable ‘x’.




```
name = input('Enter file:')
```

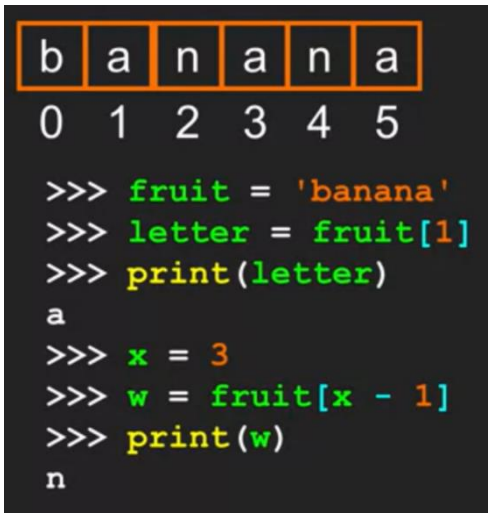
‘input’ asks the user for an input
code will read:
Enter file: [this is where the user would type something]

```
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
```

‘type’ function tells you the type of variable it is
[either string (str), integer (int) or floating point (float)]

Python cares a *lot* about how far a line is indented. If you mix tabs and spaces, you may get "indentation errors" even if everything looks fine

1 tab = 4 spaces
use spaces



index must start from 0

you will get a TB if the index is more than the number of letters in the string
i.e. if it said "letter = fruit[6]" it would give a TB

2.2: Operations

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal

Remember: "=" is used for assignment.

for comparison operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

for example, if I write: 23%7 it will be, 2
because 23/7 is 3 remainder 2

```
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

can add strings together

```
>>> print(10 / 2)
5.0
```

integer division produces floating point result

2.3: Functions

```
nam = input('Who are you?')
print('Welcome', nam)
```

```
Who are you?
Chuck
Welcome Chuck
```

program asks for an input

print statement has welcome before printing the user input

comma in the middle counts as space

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

asks for user input
'int' function turns the user typed input (which is automatically a string) into an integer

```
astr = 'Bob'
try:
    print('Hello')
    istr = int(astr)
    print('There')
except:
    istr = -1

print('Done', istr)
```

variable 'astr' is a string, and converting this into an integer will give a traceback (TB) – if you are worried the program will return a TB, put a 'try' function

if the program fails at any line within the 'try' block of code, it will jump straight to the 'except'

this function will try to convert astr into an integer, if it fails (i.e. gives a TB) it will go to the 'except' block of code

Program:

```
def thing():
    print('Hello')
    print('Fun')

thing()
print('Zip')
thing()
```

Output:

```
Hello
Fun
Zip
Hello
Fun
```

'def' is the define function code

it is a function you can define and re-use throughout your code

```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

>>>
```

'max' is a function that finds the largest value in a string (in this case it is 'w')

'min' function does the opposite which is why it prints out a space

these functions were already defined for us by python

```
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
```

can convert integer numbers in to floating point numbers

An argument is a value we pass into the function as its input when we call the function

We use arguments so we can direct the function to do different kinds of work when we call it at different times

We put the arguments in parentheses after the name of the function

```
big = max('Hello world')
```

Argument

```
>>> def greet(lang):
...     if lang == 'es':
...         print('Hola')
...     elif lang == 'fr':
...         print('Bonjour')
...     else:
...         print('Hello')
...
>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
>>>
```

in this case, 'lang' is like an alias for the argument
it is keeping it vague so when you run this function with a specific argument, the function 'greet' knows what it is looking through

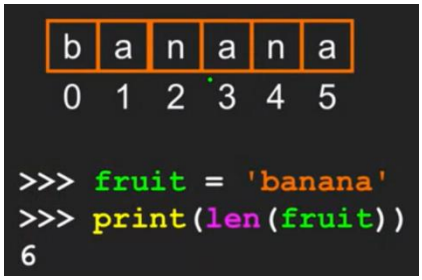
A “fruitful” function is one that produces a result (or return value)

The return statement ends the function execution and “sends back” the result of the function

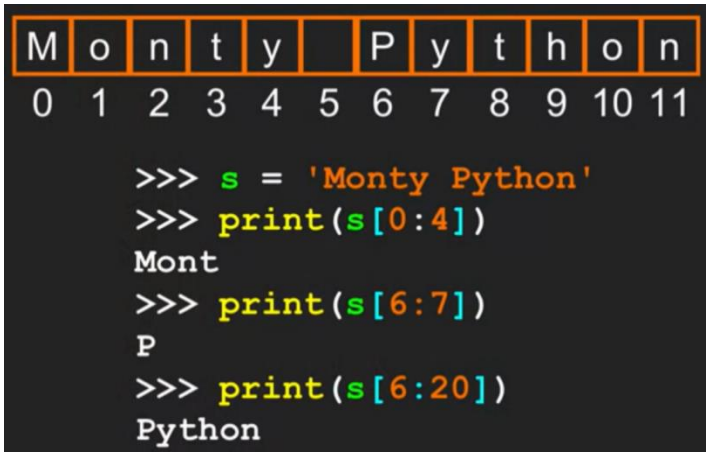
```
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
>>>
```

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)

8
```

another example of the ‘return’ function
don’t need to write ‘print(added)’ to give the answer when the function returns the value ‘added’ at the end of the definition



‘len’ function tells you the length of a string



[0:4] means starting from 0 and end before 4 (doesn’t include 4)

going over the index limit here will not give a TB

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

leaving a gap here means from the beginning

leaving a gap here means to the end

leaving gaps in both places means print from start to finish

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

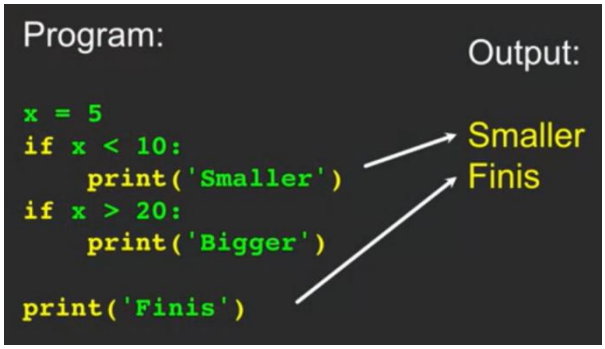
will make strings lower case (but keep the original the same)

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']

https://docs.python.org/3/library/stdtypes.html#string-methods
```

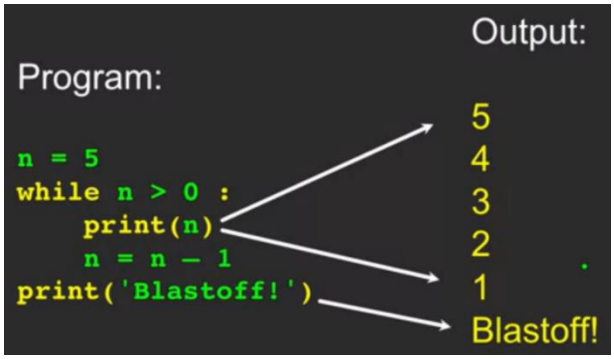
what strings are capable of

https://docs.python.org/3/library/stdtypes.html#string-methods

2.4: Loops



'if' statement is a definite conditional loop



'while' loop is an indefinite conditional loop

This loop is an infinite loop!

```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')
```

indentation is important – it tells python when the block of code/loop is finished

```
x = 4

if x > 2 :
    print('Bigger')
else :
    print('Smaller')

print 'All done'
```

two-way code with 'else' function

```
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```

multi-way conditional

elif = else + if

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

asks for user input and 'while' loop could keep this loop going forever
if we have a 'break' function, this while exit the loop (to line "print('Done!')"

The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

line[0] means the first character of the user typed string – if it is 0 then restart the iteration (i.e. from "line = input('> ')"
'break' jumps out of the loop

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

definite loop 'for'

another example

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
   print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally

Done!
```

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

counting in a loop
starts at 0 and iterates through the set of data, each time adding 1 to the count (zork)

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

summing in a loop

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25
```

finding average through loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print 'Large number',value
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

filtering through loop doesn't have to print anything

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
```

```
$ python search1.py
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

If we just want to search and know if a value was found, we use a variable that starts at False and is set to True as soon as we find what we are looking for.

boolean search
we could put a break line in once found=true to quit the code as soon as it has found its target

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

We still have a variable that is the smallest so far. The first time through the loop smallest is None, so we take the first value to be the smallest.

when finding the lowest number, instead of starting with some arbitrary value for the 'smallest' variable, I can use 'none'

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value

print('After', smallest)
```

- Python has an is operator that can be used in logical expressions
- Implies "is the same as"
- Similar to, but stronger than ==
- is not also is a logical operator

is operator ==

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
5 a
```

looping through strings

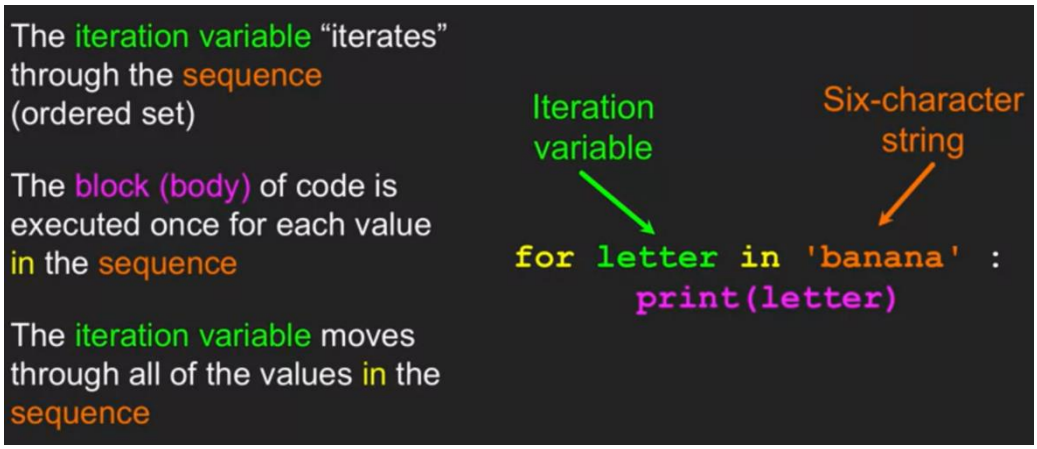
```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

```
b
a
n
a
n
a
```

this loop is much more concise – better

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

"for letter in word" basically goes through, letter by letter, the variable word – each time it see's an 'a' it will count




```python
def fizz_buzz(num):
    if num%3==0 and num%5==0:
        return 'FizzBuzz'

    elif num % 3 == 0:
        return 'Fizz'

    elif num % 5==0:
        return 'Buzz'
    else:
        return num

for n in range(1000):
    print(fizz_buzz(n))
```

define function

if number is a multiple of 3 and 5, print fizzbuzz

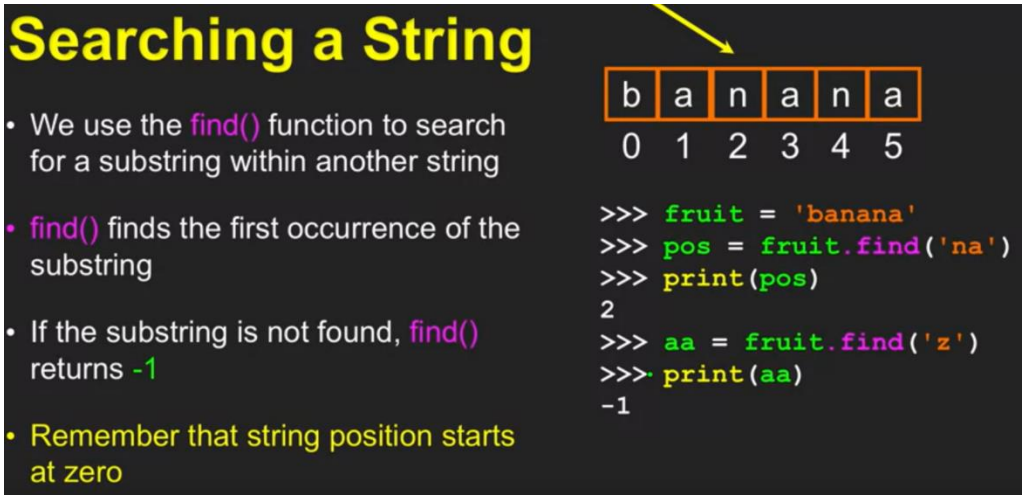
if number is only multiple of 3, print fizz

if number is only multiple of 5, print buzz

if neither, print the number

start the iteration from 1 – 1000, using the for loop

2.5: Searching strings



finding parts of string and giving the position

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob','Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HellX BXb
```

search and replace

```
>>> greet = '    Hello Bob    '
>>> greet.lstrip()
'Hello Bob    '
>>> greet.rstrip()
'    Hello Bob'
>>> greet.strip()
'Hello Bob'
```

stripping whitespace

opening a file (r = read, can be open, read, write, close)

```
                    21        31
                    ↓         ↓
 From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

finding position and printing parts of string

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

using \n to create a new line
\n only counts as one character

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

for loop goes through file (which is essentially just strings) and pulls out the word cheese as many times it occurs

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)


$ python open.py
Line Count: 132045
```

counting lines in a file

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

reading the whole file
reads file into single string with newlines (\n)

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:') :
        print(line)
```

for loop in file
finding lines that start with 'from:'

```
From: stephen.marquard@uct.ac.za\n
\n
From: louis@media.berkeley.edu\n
\n
From: zqian@umich.edu\n
\n
From: rjlowe@iupui.edu\n
\n
...
```

the output looks like this without the \n

the print statement automatically adds a newline to each line

the green \n is from the file, the yellow one has been added by the print statement

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

output

- We can strip the whitespace from the right-hand side of the string using rstrip() from the string library
- The newline is considered "white space" and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

stripping newline from printed result

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print(line)
```

same program, just using 'if NOT', then continue (i.e. by going back up to the beginning of the loop, starting with for)

We can look for a string anywhere in a line as our selection criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print(line)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

looking for string in lines in a file

```
fname = input('Enter the file name:  ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

counting lines from file input

```
fname = input('Enter the file name:  ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    quit()

count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

bad file name

the quit statement ends the program

7.2 Write a program that prompts for a file name, then opens that file and reads through the file, looking for lines of the form:

```
X-DSPAM-Confidence:    0.8475
```

Count these lines and extract the floating point values from each of the lines and compute the average of those values and produce an output as shown below. Do not use the sum() function or a variable named sum in your solution.

You can download the sample data at http://www.py4e.com/code3/mbox-short.txt when you are testing below enter **mbox-short.txt** as the file name.

```
# Use the file name mbox-short.txt as the file name
fname = input("Enter file name: ")
fh = open(fname)
count = 0
sline = 0
for line in fh:
    if line.startswith("X-DSPAM-Confidence:"):
        count = count + 1
        fline = line[21:]
        nline = float(fline)
        sline = sline + nline
    else:
        continue
kooshpa = sline/count
print("Average spam confidence:", kooshpa)
```

extracting floating point numbers and taking an average

2.6: Lists

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

square brackets indicate lists

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
```

looking inside lists

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

strings are immutable – cannot be changed (in this example we cannot change the letters, i.e. changing B to b)

however in lists we can replace numbers in the list

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
```

length of a list

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
```

range function returns list of numbers from 0 to one less than parameter length

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
    print('Happy New Year:', friend)

for i in range(len(friends)) :
    friend = friends[i]
    print('Happy New Year:', friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
```

both loops produce same result – the i one is useful for counted loops

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

concatenating (adding) strings

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

can slice lists

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

making a list from scratch
start by making an empty list

add elements using append

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
```

is something in a list?

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[1])
Joseph
```

sorting in alphabetical order

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

different functions in lists

```
total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print('Average:', average)
```

```
Enter a number: 3
Enter a number: 9
Enter a number: 5
Enter a number: done
Average: 5.66666666667
```

```
numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)
```

green method is using a list to find the average – uses less variables

however the list method needs all numbers in memory – so if there are >100,000,000 pieces of data, maybe use the first method

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
```

```
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

split function takes the spaces away and stores each word as an individual item in a list

```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
```

more than one space doesn't matter as the split function will ignore them

can select the delimiter (i.e. instead of looking for spaces you can look for semicolons and split up the string using that delimiter)

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

```
Sat
Fri
Fri
Fri
...
```

printing the days of each line after splitting

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

double split pattern – after default split, split again using @ symbol

```
fname = input("Enter file name: ")
try:
    fh = open(fname)
except:
    print("File cannot be opened")
    quit()

lst = list()
for line in fh:
    line = line.rstrip()
    sline = line.split()
    for word in sline:
        if word not in lst:
            lst.append(word)

lst.sort()
print(lst)

```

program asks for file input (with try)

first create an empty list

then by using a for loop to go through each line in the file:
- strip in the line of any whitespace
- split the line so it produces a list of each word, for example: [I, like, to, eat, poo]

using another for loop but this time to iterate through each word in the list:
- here we say if the word is not in the list we created at the beginning, add it
once added it goes back to the first for loop where it goes through the next line

sort the list alphabetically and print

```
fname = input("Enter file name: ")
try:
    fh = open(fname)
except:
    print("File cannot be opened you little goomba")
    quit()

count = 0
lst = list()
for line in fh:
    line = line.rstrip()
    if line.startswith("From "):
        sline = line.split()
        count = count + 1
        print(sline[1])

print("There were", count,"lines in the file with From as the first word")
```

reads each line, strips it

if the line starts with from, it splits the line into words, adds to the count and prints the second word in the split line list

2.7: Dictionaries

- Dictionaries have different names in different languages
  - Associative Arrays - Perl / PHP
  - Properties or Map or HashMap - Java
  - Property Bag - C# / .Net

dictionaries are the second data structure we learn after lists
dictionaries also store multiple values in them however they are not ordered like lists and each item you add to the dictionary needs a label (a key)

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

dictionaries are like unsorted bags of items

here we make a new empty dictionary called purse

with the first assignment statement we use the index operator to label the value 12 under the key 'money' – so the value is 12, the key is 'money'

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

lists and dictionaries are similar (both mutable)
dictionaries use curly brackets
lists use positions to retrieve values however dictionaries use keys/labels

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{}
```

can make a dictionary with values in like the first line – has to go key then value, key-value…
can make an empty dictionary with { }

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

can't open a key that isn't there

we can check if the key is present in the dictionary though

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```

goes through names, any new names you add to the dictionary with a count of 1, any existing names it updates the count by adding one

```
x = counts.get(name, 0)


{'csev': 2, 'zqian': 1, 'cwen': 2}
```

get function checks if this key is in the dictionary
the 'name' is where the key goes, and the zero is a default, meaning if you put a key that does not exist in the dictionary you will get a zero back
if I put counts.get(csev, 0) in I should get 2 back

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

different way of tallying up by using the get function – if the name doesn't exist it will return the default value of 0, and then add 1

```
counts = dict()
line = input('Enter a line of text:')
words = line.split()

print('Words:', words)
print('Counting...')

for word in words:
    counts[word] = counts.get(word,0) + 1
print('Counts', counts)
```

```
python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car' 'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and' 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the', 'car']
Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}
```

splits each line up into list of words, counts each one, adding new ones to the dictionary and adding one to existing ones

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print(key, counts[key])
...
jan 100
chuck 1
fred 42
```

loop through dictionaries actually loops through keys, if you want the value attached, you must use the index operator [ ]

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
```

can convert dic to list
can print just keys/value
can print them together in a tuple (items)

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

```
jan 100
chuck 1
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

can actually iterate through two variables (i.e. here the key and the value)

remember the items is the key and value together

helpful to rename aaa to k (meaning key) and bbb to v (meaning value)

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

- read through each line and split the words
- read through the words and create a tally for each word for its occurrence
- then find the biggest count, and therefore find the biggest word attached to that count
- “if bigcount is None” is talking about the first iteration, every iteration after that, if the count is bigger than the stored value, the stored value is updated to the larger count

sent the greatest number of mail messages. The program looks for 'From ' lines and takes the second word of those lines as the person who sent the mail. The program creates a Python dictionary that maps the sender's mail address to a count of the number of times they appear in the file. After the dictionary is produced, the program reads through the dictionary using a maximum loop to find the most prolific committer.

Check Code | Reset Code | Grade updated on server.

```
fname = input("Enter a file name: ")
try:
    fhand = open(fname)
except:
    quit

counts = dict()
for line in fhand:
    if line.startswith("From "):
        words = line.split()
        email = words[1]
        counts[email] = counts.get(email,0) + 1

bigcount = None
bigname = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigname = word
        bigcount = count
```

```
hand = open(fname)

di = dict()
for lin in hand:
    lin = lin.rstrip()
    # print(lin)
    wds = lin.split()
    # print(wds)
    for w in wds:
        if w in di :
            di[w] = di[w] + 1
        else:
            di[w] = 1
            print('**NEW**')
        print(w,di[w])
```

reads through each line, strips and splits and puts the list of words in 'wds' variable

looking through each word (labelled 'w') in the list ('for w in wds')

'if w in di' means if the individual word in the list appears in the dictionary, add one to its value (value denoted by index notation [ ])

If not in the dictionary attach a number one to it and obviously this will add the word to the dictionary (di[w] means you've put the key in already, which is the word (w) and then attached the 1 value to it)

the print statement prints the item (both key and value)

2.8: Tuples

```
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print(x[2])
Joseph
>>> y = ( 1, 9, 2 )
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```

```
>>> for iter in y:
...     print(iter)
...
1
9
2
>>>
```

tuples are like lists… they use less memory

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>>[9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback:'str'
object does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple'
object does
not support item
Assignment
>>>
```

they are immutable you cannot append, sort…

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```

you can put tuples on either side of an assignment statement, they will map each thing in correspondence

i.e. x = 4, y = fred
a = 99, b = 98

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

d.items gives us a list of tuples e.g. (‘csev’, 2)

for (k,v) in d.items basically iterates through the list of tuples, it goes to the first tuple and pulls k (meaning the key) and then pulls v (value) which is why we get first:
csev 2

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ('Jones', 'Sam')
True
>>> ( 'Jones', 'Sally') > ('Adams', 'Sam')
True
```

checks order of first, if different it doesn't check the rest

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
```

sorts tuples based on keys only as you cannot have a duplicate key

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

prints the key and value of the dictionary but it sorts the tuples first (key order) which is why it prints out in order

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...         tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

what if I wanted to sort by value:
create an empty list and append it with the items (or tuples) from the dictionary except we do the value first then the key
then sort this list
reverse=True means it goes from big to small now (rather than small to big)

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = list()
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)

for val, key in lst[:10] :
    print(key, val)
```

The top 10 most common words

fill dictionary up w/ words, keep a tally, make a list, look up the tuples from the dictionary, change the order of key-value-pairs and update the list, sort the list, from big to small, list stops on the 9th number

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )

[(1, 'b'), (10, 'a'), (22, 'c')]
```

same program, less work

```
name = input("Enter file:")
fhand = open(name)

hlist = dict()
for line in fhand:
    if line.startswith("From "):
        words = line.split()
        time = words[5]
        tsplit = time.split(":")
        hour = tsplit[0]
        hlist[hour] = hlist.get(hour,0) + 1

hsort = sorted(hlist.items())
for k,v in sorted(hlist.items()):
    print(k,v)
```

read the file, create a dictionary, read line by line, if it starts with from, split the line into words, look for the 6th word (which is the time), then split the time (via colons), take the 1st word (which is the hour), then tally all the hours in the dictionary, sort the dictionary in numerical order, then print the hours and the tally side by side

2.9: Regular Expressions

```
^        Matches the beginning of a line
$        Matches the end of the line
.        Matches any character
\s       Matches whitespace
\S       Matches any non-whitespace character
*        Repeats a character zero or more times
*?       Repeats a character zero or more times (non-greedy)
+        Repeats a character one or more times
+?       Repeats a character one or more times (non-greedy)
[aeiou]  Matches a single character in the listed set
[^XYZ]   Matches a single character not in the listed set
[a-z0-9] The set of characters can include a range
(        Indicates where string extraction is to start
)        Indicates where string extraction is to end
```

regular expressions

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print(line)
```

does the same thing, on right uses regular expressions, we first have to import from the library though

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print(line)
```

this character ^ tells us look at the beginning of each line for the letter F

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

^X.*:

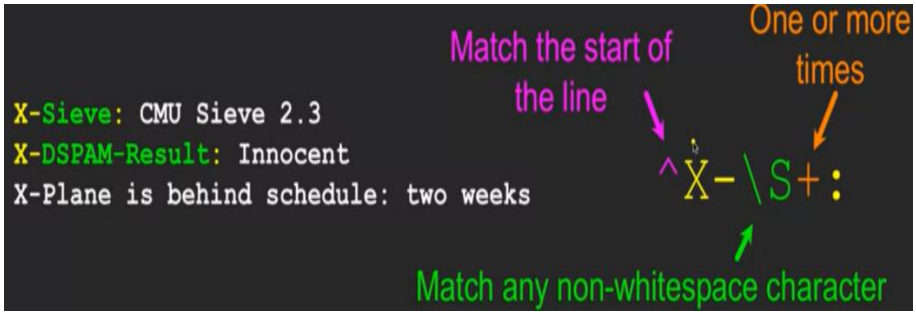
^X means starts with X followed by a colon,

. means any character

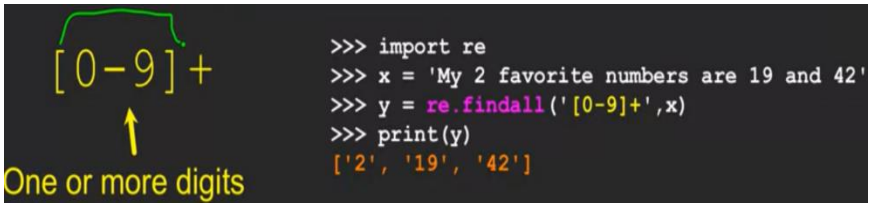
* means any number of times

so ^X.*: would return these strings

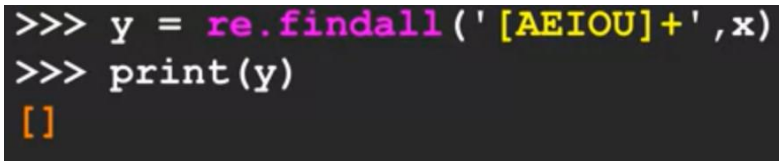
: means followed by a colon



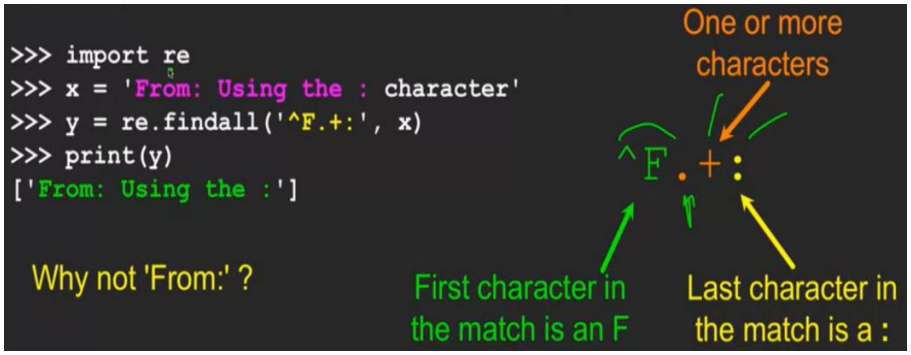
\S means a character that isn't a space one or more times, so if it reaches a space before the colon, it does not include it, hence why the last line has not been retrieved



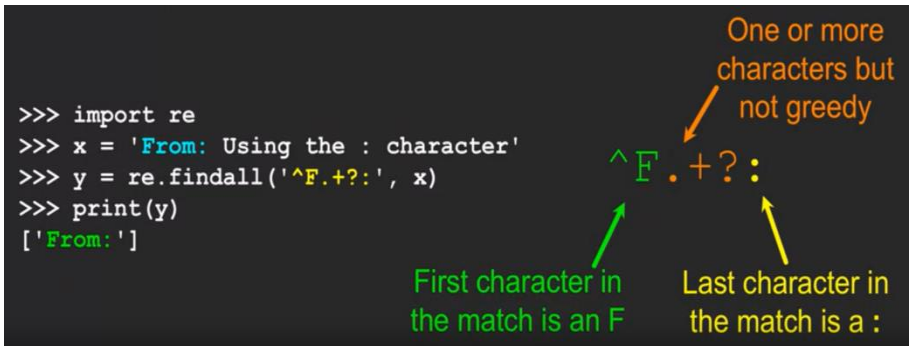
[0-9] means all digits, + means one or more
findall function searches through and returns a list of strings



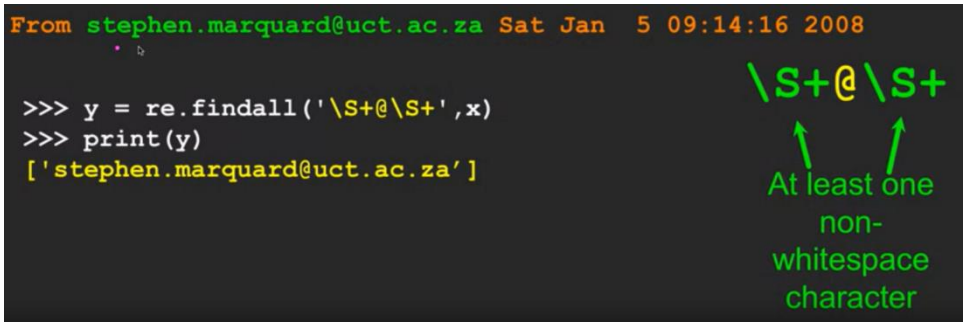
asking if there are one or more uppercase vowels
it returns an empty list



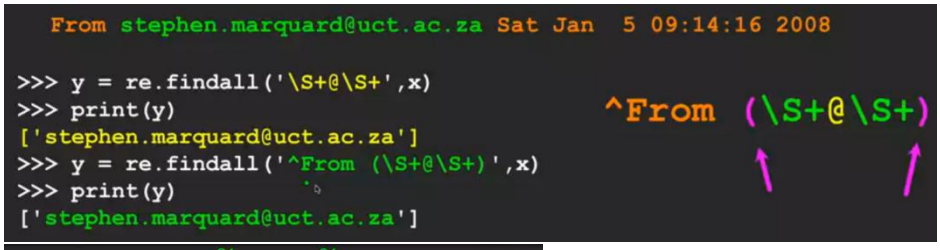
starts with F, any character one or more times, ends with colon
it takes the biggest possible version, so instead of stopping at From:, it stops at From: Using the :
to fix this we could've used the \S to mean characters not including whitespaces
OR…



…use a question mark – it stops this find function from being 'greedy' and takes the shorter string

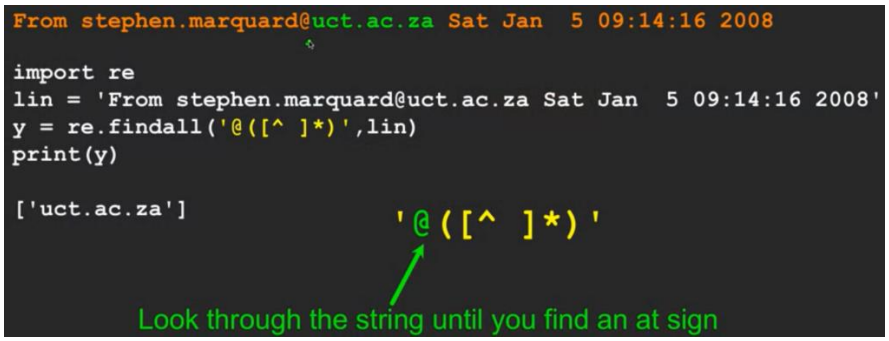


starts with character, one or more times, then @ symbol, then ends with any character that isn't a space one or more times

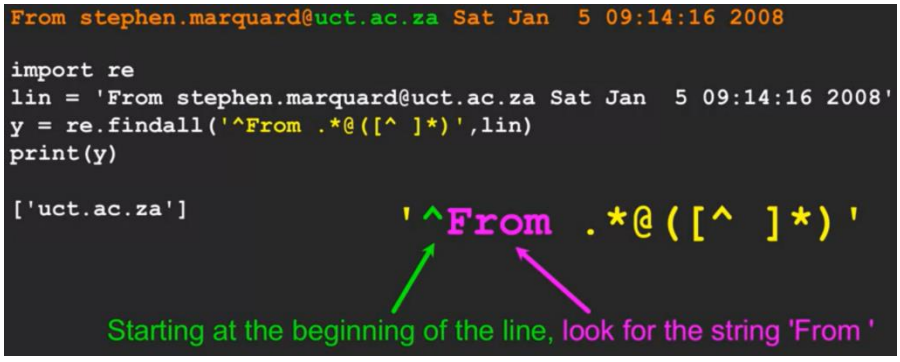


look for strings starting with from but the ( ) means it does not extract that part – only extracts the ( )

they both do the same thing, left is using position, right is using split



this one uses re
look through until you see @, extract the after that: the [^] means NOT, so if after the @ there is a space, do not extract (i.e. anything but a space), * means any number of times



line starting with 'From ' followed by any character (any number of times) until an @, extract any character as long as it's not a space

```
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 :  continue
    num = float(stuff[0])
    numlist.append(num)
print('Maximum:', max(numlist))
```

```
X-DSPAM-Confidence:  0.8475
```

```
python ds.py
Maximum: 0.9907
```

line beginning with …., extract any floating point number (because of the point) one or more times

if the length of the list is not 1, skip that line and go to for loop above

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9.]+',x)
>>> print(y)
['$10.00']
```

At least one or more

\$[0-9.]+

A real dollar sign

A digit or period

\$ is to actually write the dollar sign

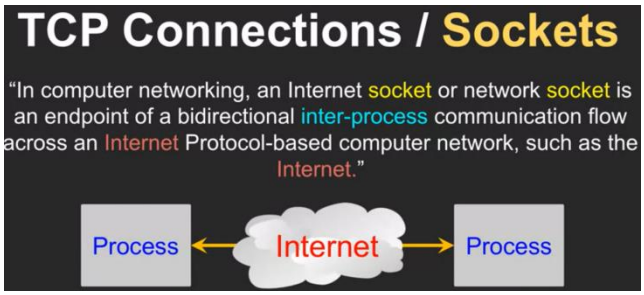
KEY:

+ means one or more times . means any character
* means zero or more times \S means any character except space

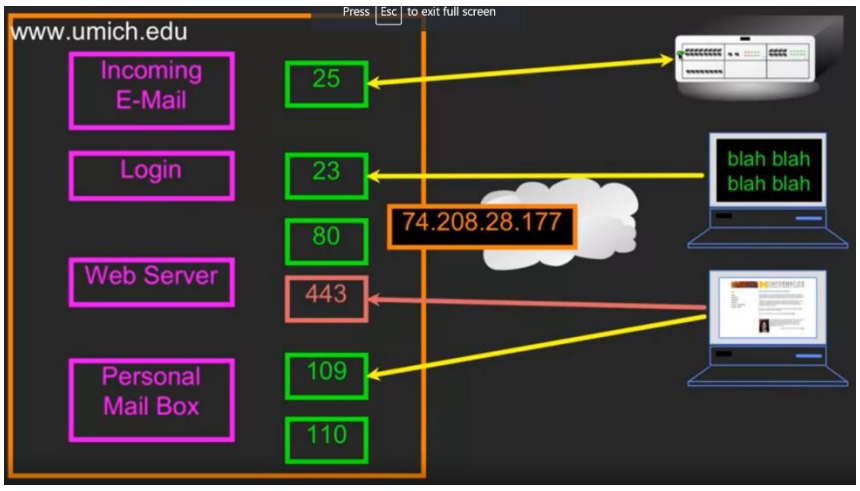
```
import re
fhand = open('regex_sum_1307698.txt')
numlist = list()
for line in fhand:
    fnum = re.findall('[0-9]+', line)
    for thing in fnum:
        num = int(thing)
        numlist.append(num)
x = 0
for digit in numlist:
    x = x + digit
print(x)
```

read through file and create an empty list
reading through each line, find all integers and put into empty regular expression list (will return strings in a list)
for each item in the list, make it an integer
add each integer to the empty list at the beginning
start the count at 0
for every number in the list add it to get the sum of all numbers from the text

2.10: Sockets



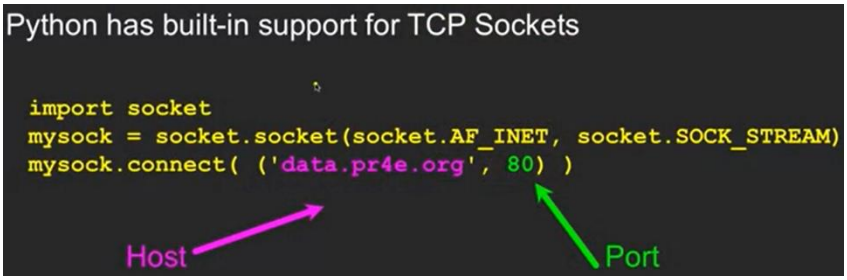
called sockets



establish a socket to talk to a server (which has a name and number)

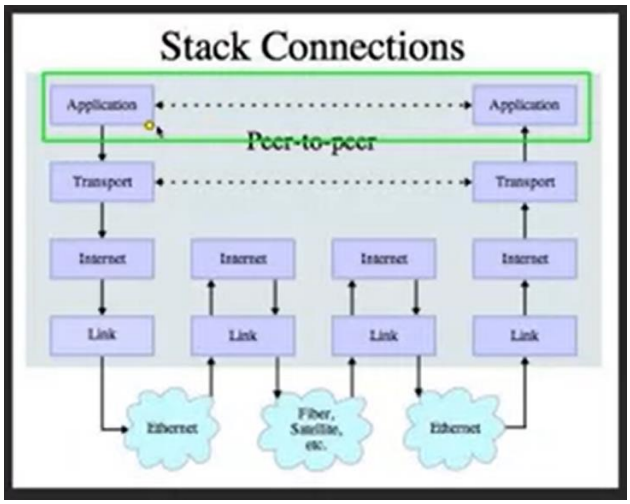
different applications on the server have different ports

web servers usually on port 80



import socket library
create a socket
connect to said socket

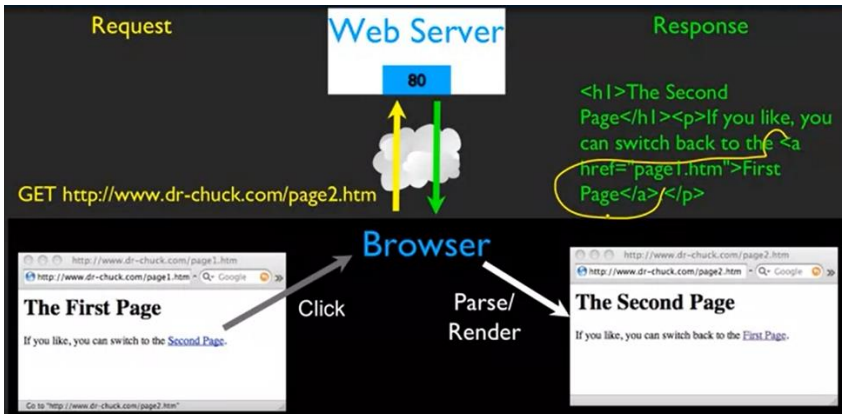
URL – Uniform Resource Locator



made a socket in the transport layer, time to connect to a port and send our program in the application layer to the other side in the application layer

depending on which server we connect to determines the rules needed to start sending data – these rules are called protocols

http is the HyperText Transfer Protocol – set of rules that allow browsers to retrieve web documents from servers over the internet

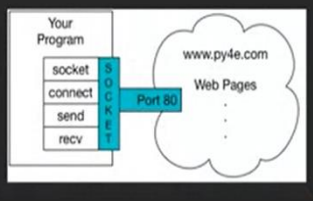


click the link, communicates with browser, browser makes a socket connection on port 80 and sends a get request to port 80 on web server, web server parses the request, makes a response (HTML form) and sends it back to browser, browser renders the response and shows the link

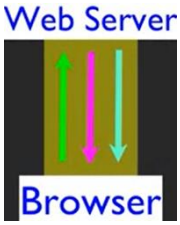
```
$ telnet data.pr4e.org 80
Trying 74.208.28.177...
Connected to data.pr4e.org.Escape character is '^]'.
GET http://data.pr4e.org/page1.htm HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 08 Jan 2015 01:57:52 GMT
Last-Modified: Sun, 19 Jan 2014 14:25:43 GMT
Connection: close
Content-Type: text/html

<h1>The First Page</h1>
<p>If you like, you can switch to
the <a href="http://data.pr4e.org/page2.htm">Second
Page</a>.</p>
Connection closed by foreign host.
```

telnet is a software that allows insecure connections to webservers
first connect to the host and port (yellow)
then send a get request to the document on the host, through the URL (green)
you will get back headers, which are just metadata – info about the file (pink)
you will receive the file and the connection will be closed (blue)

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```

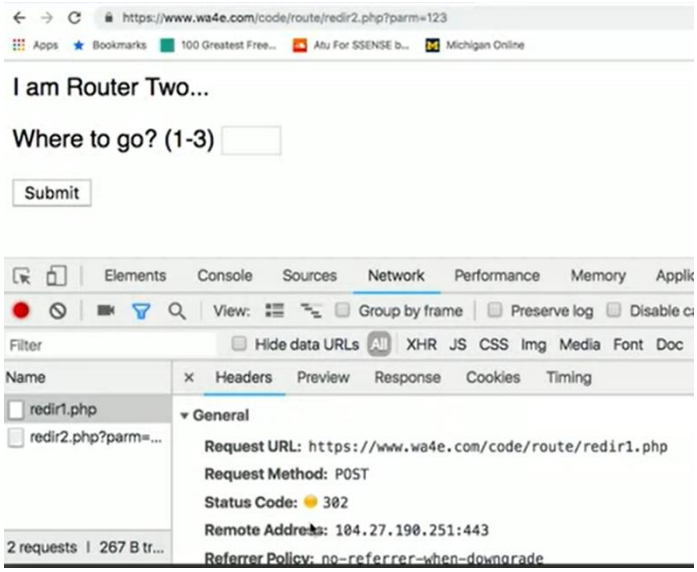


import socket, create socket, connect socket to server (host and port 80), create get command (have to prepare it for sending which is why we encode it at the end), then send command through socket
create while loop saying receive 512 characters, if there are no characters in the data, break the file/transmission, otherwise print it (via decode) and then close the socket (takes resources from both sending and receiving computers)



encode function turns pythons Unicode into UTF-8

get requests at end of URL can also be called query string parameters (found in source code)



looking at this url, using the developer code, on 'network' tab

can get different status codes:
- 200 means ok, good, you asked for something and the server gave it
- 404 means not found, error
- 302 means I could not give you something based on your get request but here is a redirect link (this redirect normally has 200 status code)

here can find what content type each page is

In a client-server application on the web using sockets, which must come up first?

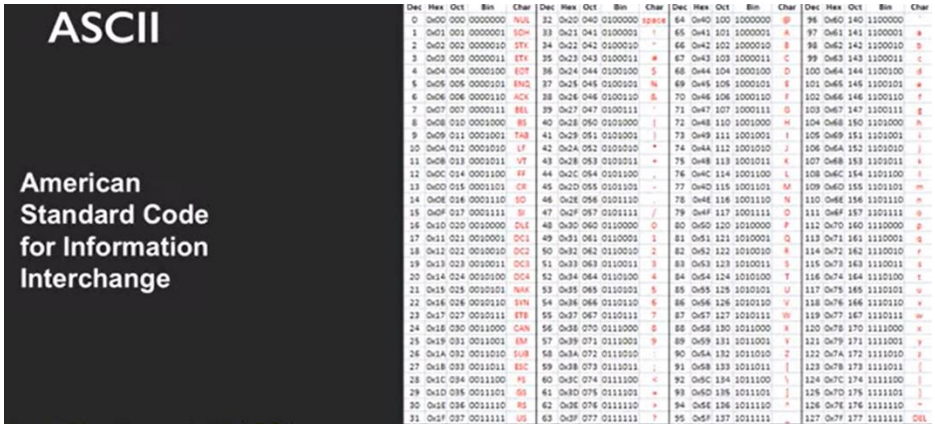
- (●) server
- ( ) client
- ( ) it does not matter

A GET REQUEST

When you click on an anchor tag in a web page like below, what HTTP request is sent to the server?

```
1   <p>Please click <a href="page1.htm">here</a>.</p>
2
```

2.11: Encoding/Decoding



the mapping we commonly use is ASCII

it maps characters to numbers

for example, H is 72

this is how computers used to store data

can use ord('H') to find this value (ord means ordinal)

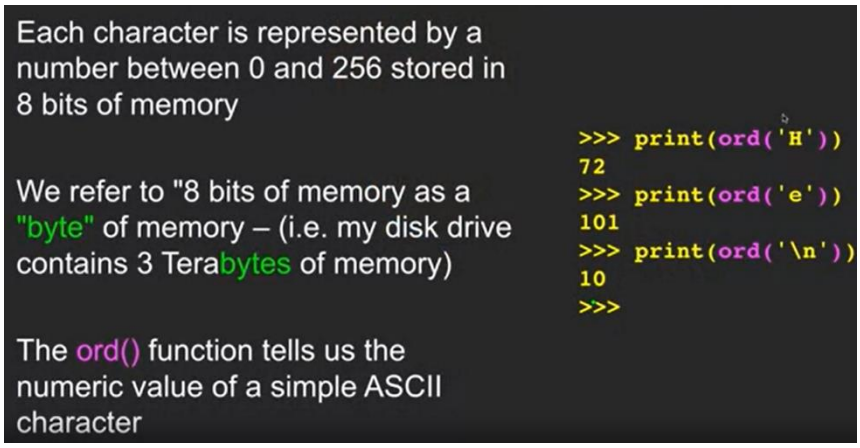
each character represented by number between 0 and 256 stored in 8 bits of memory
8 bits of memory is called a byte

hex means hexadecimal, octodecimal and binary

couldn't talk to other computers this way so they invented Unicode

ordinal tells us the numeric value of the ASCII character

```
>>> print(ord('H'))
72
```



each character stored in 8 bits which is a byte of memory

\n means new line

ASCII not practical, cannot communicate different languages

UTF-8 is the best – variable byte size (1-4) so maps onto ASCII nicely

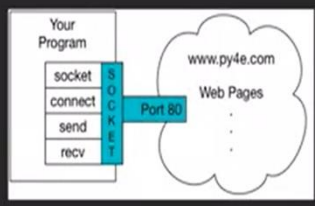
```
Python 3.5.1
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
```

byte strings type is bytes

unicode string type are strings

Python written in Unicode

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```



talking to a network socket we receive and send bytes, however when working in python we use Unicode

encoding and decoding almost all of the time change from UTF-8 to Unicode, so no need to specify (I.e. We can leave () blank next to encode or decode)

```
Python 2.7.10
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3.5.1
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

In Python 3, all strings are Unicode

in python 2 you had to convert strings into Unicode, however python 3 all strings are Unicode (not utf-8)

```
Python 3.5.1
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
```

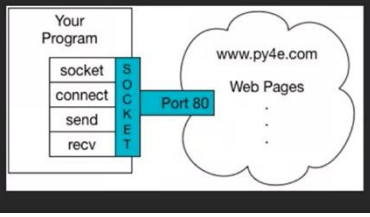
in python 2 this would have returned a string but in python 3 it is a byte – it means raw, unencoded data, could be utf-8, 16, 32, ASCII, we don't know the encoding

```
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    mystring = data.decode()
    print(mystring)
```

data.decode assumes utf-8 but it will find out what encoding and convert to Unicode

data is bytes (because we don', mystring is unicode

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```



a http request

we are sending data, so we must encode it (turn it into bytes)

2.12: URL's

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())
```

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

can open and read the Web document just from python by using urllib

Just make sure to decode as it will be in UTF-8

Pull data - particularly social data - who links to who?

Get your own data back out of some system that has no "export capability"

Monitor a site for new information

Spider the web to make a database for a search engine

web scraping

program that pretends to be a browser, retrieves and extracts info from web pages

these are the reasons you may web scrape

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup

url = input('Enter - ')
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html, 'html.parser')

tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

import BeautifulSoup program
enter url, open url and read it whole, use BeautifulSoup to parse the page (basically sorts out the ugly side of web form of HTML)
retrieve the anchor tags (i.e. looking for 'a')
print the tags (i.e. all the links from the webpage without the ugly formatting)

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

this is an anchor tag

```python
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

this block of code (ctx) and also slight addition in the html line is the same code earlier adjusted for https (secure websites, with SSL)

Which of the following Python data structures is most similar to the value returned in this line of Python:

```
1   x = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
```

- ○ list
- ○ regular expression
- ◉ file handle
- ○ dictionary
- ○ socket

Which of the following regular expressions would extract the URL from this line of HTML:

```
1   <p>Please click <a href="http://www.dr-chuck.com">here</a></p>
```

- ◉ href="(.+)"
- ○ href=".+"
- ○ http://.*
- ○ <.*>

```python
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl
import re

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).rea
soup = BeautifulSoup(html, 'html.parser')

tags = soup('span')
for tag in tags:
    #y = re.findall('\S+>([0-9+])', tag)
    #print(y)
    print('Tag:', tag)
    print('URL:', tag.get('href', None))
    print('Contents:', tag.contents[0])
    print('Attrs:', tag.attrs)
```

this program asks for a URL, parses it and finds all 'span' tags, prints the tag, the URL's (if there are any), the contents (number) and whatever that is at the end?

```
C:\Users\fayaz\OneDrive\Desktop\Code>webscrape1.py
Enter -  http://py4e-data.dr-chuck.net/comments_42.html
Tag: <span class="comments">97</span>
URL: None
Contents: 97
Attrs: {'class': ['comments']}
Tag: <span class="comments">97</span>
URL: None
Contents: 97
Attrs: {'class': ['comments']}
Tag: <span class="comments">90</span>
URL: None
Contents: 90
Attrs: {'class': ['comments']}
Tag: <span class="comments">90</span>
URL: None
Contents: 90
Attrs: {'class': ['comments']}
```

```python
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl
import re

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

total = 0
count = 0
tags = soup('span')
for tag in tags:
    #print('Tag:', tag)
    #print('URL:', tag.get('href', None))
    #print('Attrs:', tag.attrs)
    count = count + 1
    val = tag.contents[0]
    num = int(val)
    total = total + num

print('Count', count)
print('Sum', total)
```

open url, parse it, start a total and count, look for span tags, for each tag extract the numbers, convert to an integer, add to the total, print

```
url = input('Enter - ')

count = int(input('Enter count: '))
pos = int(input('Enter position: '))

print('Retrieving: ', url)

index = 0

lst=list()
while index < count:
    html = urllib.request.urlopen(url, context=ctx).read()
    soup = BeautifulSoup(html, 'html.parser')
    tags = soup('a')
    for tag in tags:
        x = tag.get('href', None)
        lst.append(x)
    url = lst[pos-1]
    print('Retrieving: ', url)
    index = index + 1
    lst.clear()

name = re.findall('([A-Za-z]+).html$', url)
print(name)
```

after importing everything

ask for count (number of times we will repeat the web crawl) and position (which url to open)

start the index at 0, start a new list
while loop basically means, while the index is smaller than the count keep going, i.e. every loop we add one to the index, when the index is bigger than the count then it stops repeating

open the url, parse, find the anchor tags, start the for loop for each tag and extract the urls (from tag.get(href)), add them to a list exit the loop and take the url of the position we asked for in the beginning (using same variable url ,so replacing the old url), add one to the index and clear the list

```
C:\Users\fayaz\OneDrive\Desktop\Code>followlink.py
Enter -  http://py4e-data.dr-chuck.net/known_by_Vanni.html
Enter count: 7
Enter position: 18
Retrieving:   http://py4e-data.dr-chuck.net/known_by_Vanni.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Anabella.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Chelsi.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Architha.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Rohan.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Krista.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Ailie.html
Retrieving:  http://py4e-data.dr-chuck.net/known_by_Silvana.html
['Silvana']
```

now we go back to the while loop, here the index is now 1 so it will repeat again, however this time we have a new url from before which opens, parses and repeats again, adding all the links in to an empty list again and taking the same position this keeps repeating until it has looped the same amount as the count

last part is a regular expression for extracting the name from the url, here we see it looks for all letters, upper and lower case and extracts that, but it must end in ".html"

```python
import urllib.request, urllib.parse, urllib.error
import requests
from bs4 import BeautifulSoup
import ssl
import re

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = 'https://www.imdb.com/list/ls071265847/'

html = urllib.request.Request(url, None, headers)
htmltext = urllib.request.urlopen(html).read()
soup = BeautifulSoup(htmltext, 'html.parser')

for movies in soup.findAll('h3', class_='lister-item-header'):
    order = movies.find('span').contents[0]
    title = movies.find('a').contents[0]
    year = movies.find('span', class_='lister-item-year').contents[0]
    #Link = movies.find('a')['href'])

    print(order + ' ' + title + ' ' + year)
```

project: imdb list

create variable headers (allows python terminal to act like my computer so should avoid 400/403 url error)

html variable can see uses this header

loop says for every h3 tag you find, but also must contain the class ‘lister-item-header’, store it in the variable movies

```
<h3 class="lister-item-header">
<span class="lister-item-index unbold text-primary">93.</span>
<a href="/title/tt0106611/?ref_=ttls_li_tt">Cool Runnings</a>
<span class="lister-item-year text-muted unbold">(1993)</span>
</h3>
None
```

this is what movies look like (if I do print(movies))

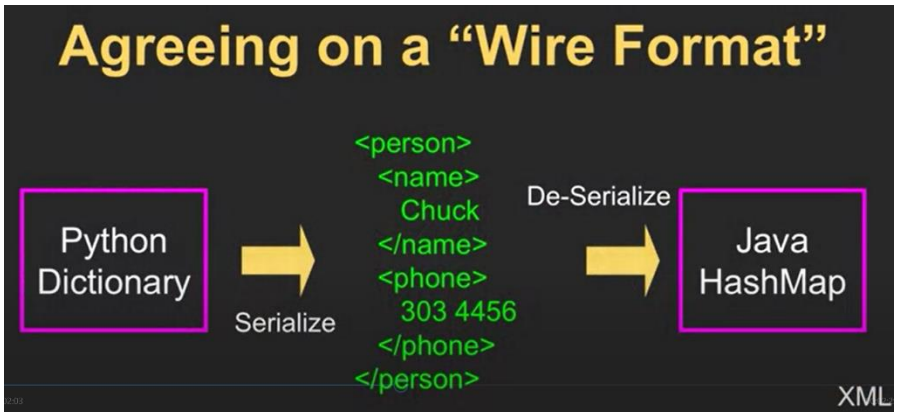
it contains what I want alongwith all the html code

```
1. The Twilight Zone (1959–1964)
2. Big Trouble in Little China (1986)
3. American Dad! (2005– )
4. Family Guy (1999– )
5. The Cleveland Show (2009–2013)
6. The League (I) (2009–2015)
7. Trailer Park Boys (2001–2018)
8. Cosmos (2014)
9. Team America: World Police (2004)
10. 30 Rock (2006–2013)
```

to get this result, look at the loop code

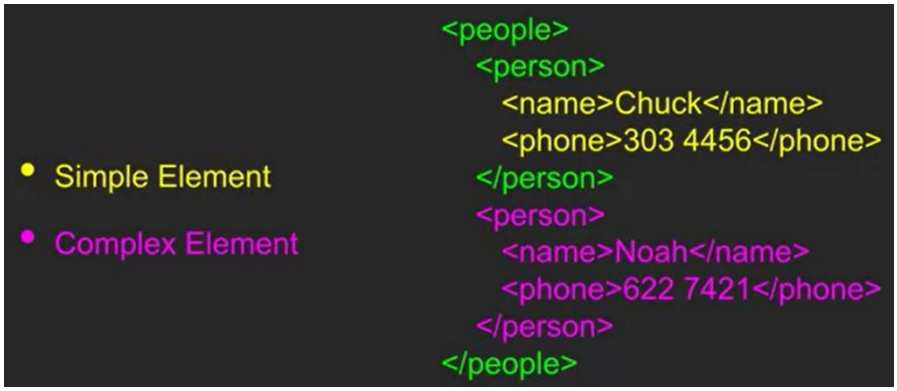
if I wanted to print the link also I can use that commented out line under ‘year…’

2.13: XML



to send things over the net, need to serialize the data into a wire format, here the wire format used is XML (extensible markup language)

can also use JSON



xml uses nodes/elements

simple element means no nesting

complex elements usually involve nesting

doesn't need indenting

Primary purpose is to help information systems share structured data

It started as a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible

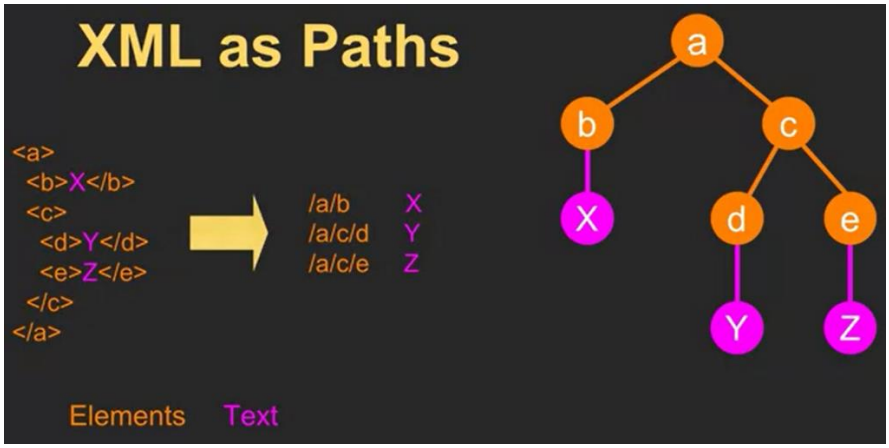
end tags have slash in it

self closing tags have empty text content node

Tags indicate the beginning and ending of elements

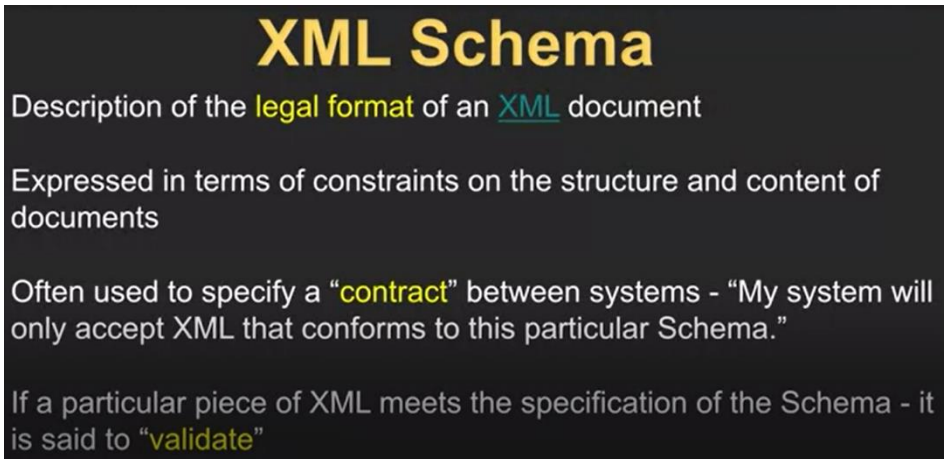
Attributes - Keyword/value pairs on the opening tag of XML

Serialize / De-Serialize - Convert data in one program into a common format that can be stored and/or transmitted between systems in a programming language-independent manner



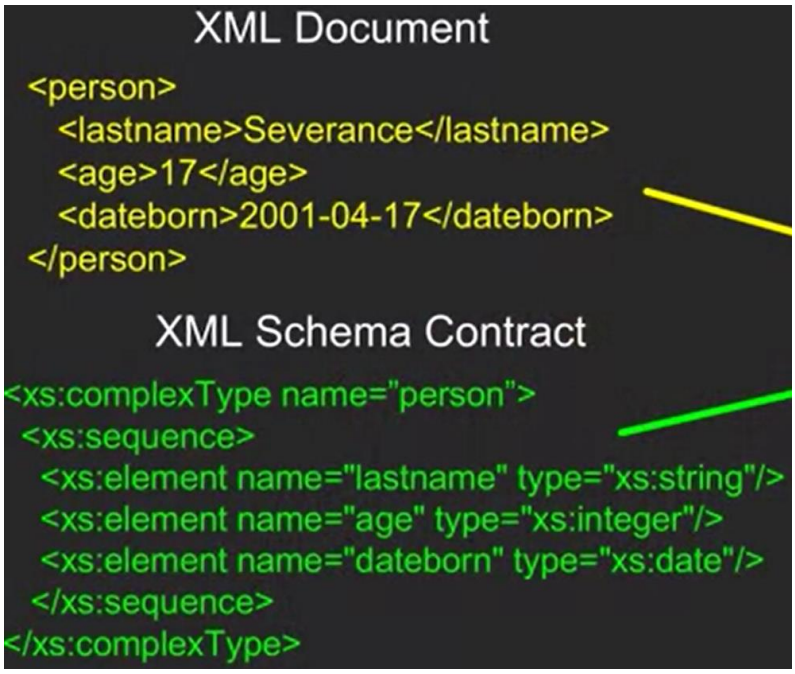
a little bit like file directories

can think of xml like a tree or with paths



xml schema is a contract to determine if good code was sent and good code was received

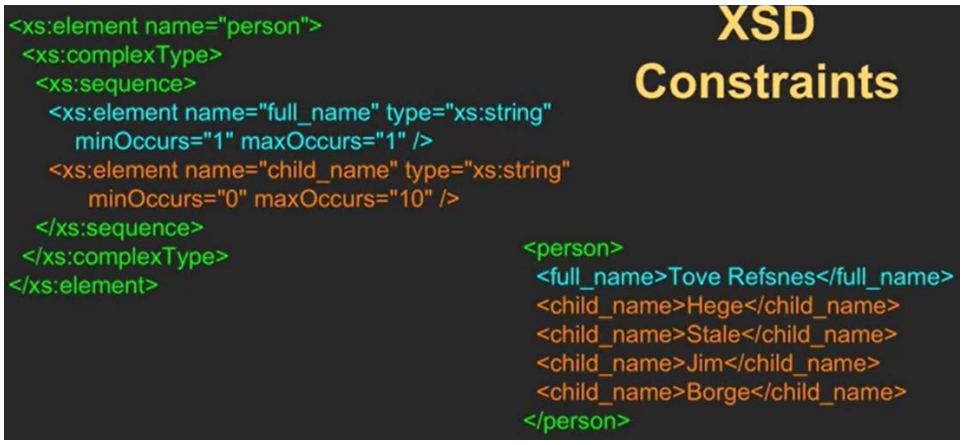
legal contract telling which system is messed up



xml schema contract looks for a complex node/element, with a person type
checks the sequence includes, elements such as last name/age/dateborn all in the correct type (e.g. string/integer etc)

if the contract and document match, it will be validated

schema normally in file format .xsd (XML W3C version)



min occurs 1 max occurs 1 means can only happen once

min occurs 0 max occurs 10, which is why we can see the orange elements 4 times, it will be validated

xsd data types

iso 8601 date/time format

ignores time zones

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string" />
        <xs:element name="House" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="Town" type="xs:string" />
        <xs:element minOccurs="0" name="County" type="xs:string" />
        <xs:element name="PostCode" type="xs:string" />
        <xs:element name="Country">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FR" />
              <xs:enumeration value="DE" />
              <xs:enumeration value="ES" />
              <xs:enumeration value="UK" />
              <xs:enumeration value="US" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<Address
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

left is xsd right is xml
first line of xsd just making sure it is in the right format, then comes the schema contract, followed by open tag with its key/value pairs, then our sequence: have all of the elements, the county has minOccurs=0 so it does not have to be present, the country element has a restriction on it, it has to be either FR, DE, ES, UK or US - as the xml document contains UK, this will validate

```
import xml.etree.ElementTree as ET
data = '''<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print('Name:',tree.find('name').text)
print('Attr:',tree.find('email').get('hide'))
```

import elementtree and then use ‘as’ to make an alias for it
‘’’ indicates a multi-line string – so the string starts before the person tag and ends after the person tag
fromstring takes the string and makes a nice tree (as seen before)
tree.find(‘name’) gives you the whole element, i.e. <name>Chuck</name>, but the .text part only returns Chuck
next line means, go find me the attribute ‘hide’ in the email element – returns yes

```
import xml.etree.ElementTree as ET
input = '''<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
        </user>
    </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))
for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

lst = stuff.findall(‘users/user’) gives us all users in a list – not a list of strings, a list of tags/ objects/trees

prints the length of the list

for loop goes through each tree in the list
for each time it prints the name, id text and user number (e.g. 2 or 7)

```
<xs:element name="item" maxOccurs="unbounded">
```

can happen as many times as you want

```python
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl
import xml.etree.ElementTree as ET

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
print('Retrieving: ', url)

sum = 0
total = 0

url = urllib.request.urlopen(url, context=ctx).read()
tree = ET.fromstring(url)
lst = tree.findall('comments/comment')
for num in lst:
    total = total + 1
    count = int(num.find('count').text)
    #count = tree.findall('.//count')
    sum = count + sum

#print('Retrieved', , 'characters')
print('Count: ', total)
print('Sum: ', sum)
```

```
<commentinfo>
  <note>This file contains
▼<comments>
  ▼<comment>
      <name>Romina</name>
      <count>97</count>
    </comment>
  ▼<comment>
      <name>Laurie</name>
      <count>97</count>
    </comment>
  ▼<comment>
      <name>Bayli</name>
      <count>90</count>
    </comment>
  ▼<comment>
      <name>Siyona</name>
      <count>90</count>
    </comment>
  ▼<comment>
      <name>Taisha</name>
      <count>88</count>
    </comment>
```

```
C:\Users\fayaz\OneDrive\Desktop\Code>xmlcount.py
Enter -  http://py4e-data.dr-chuck.net/comments_1307702.xml
Retrieving:   http://py4e-data.dr-chuck.net/comments_1307702.xml
Count:  50
Sum:  2901
```

left is code, right is sample of the url, bottom right is the output
open the url, read it, use ElementTree to create a tree of info, create a list of trees (as there is a parent node comments, and within that node are children nodes ‘comment’, which contain our useful info), going through each item in the list using a for loop, create a running count of the loop alongside a sum of all the comment counts (by converting the string extracted from the find(‘count’) line into an integer)

2.14: JSON

```
import json
data = '''{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}'''

info = json.loads(data)
print('Name:',info["name"])
print('Hide:',info["email"]["hide"])
```

json very closely related to javascript – one of the most common serialization format

json represents data as nested lists and dictionaries - {} in javascript are called objects but behave very similarly to dictionaries

in phone there is a child dictionary, or a dictionary in a dictionary

loads = load from string

info will be a python dictionary – so we use the [] notation

use [“name”] and we get Chuck
the next line takes the dictionary info, looks for email and because that is also a dictionary we then look at hide, to get yes

```
import json
input = '''[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''

info = json.loads(input)
print('User count:', len(info))
for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])
```

because the string uses a square bracket it is a list, not a dictionary – essentially we will get a list of 2 dictionaries (as we have 2 id dictionaries)

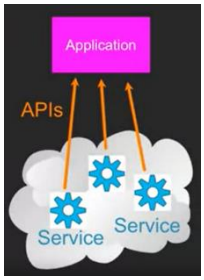
info will be a list now

that means we can use len(info)

we can use for loops to iterate through the list (will be iterating through the dictionaries in the list)

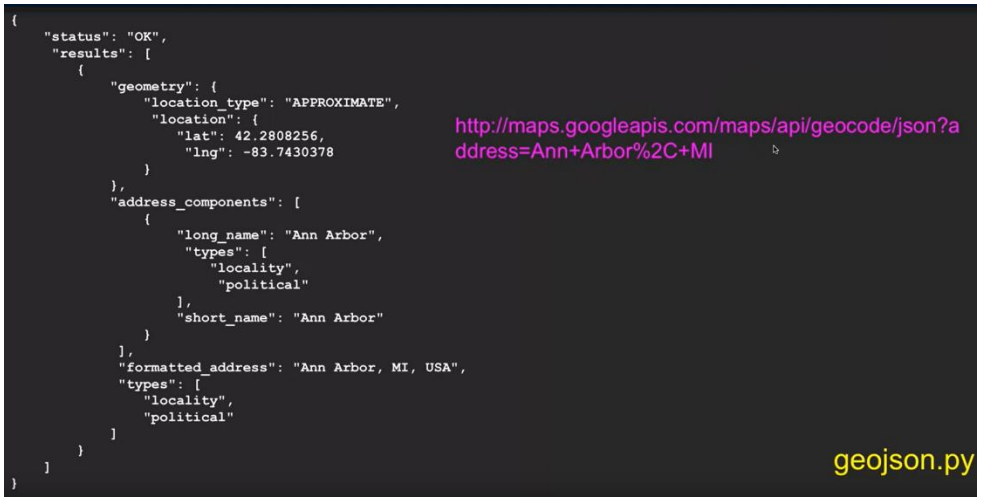
to print elements of the dictionary we must use the []

2.15: API



think of a flight website, they link car rentals and hotels – linking services

services publish the 'rules' applications must follow to make use of this service (API – Application Programming Interfaces)



- after the question mark is our parameter, address =

- + is space, % is comma

- Ann Arbor part is as if we typed this in the google search bar but instead we're talking to an API through url encoding

if you put this into a browser you will get the json on the left - this json code is a dictionary, then a list inside the dictionary (results, indicated by square brackets), followed by more dictionaries inside dictionaries

if you don't type an address, it breaks the while loop

the encode turns 'Ann Arbor, MI' into Ann+Arbor%2C+MI, it adds to the old api url

uh will load the url, data will actually open it, decode means it turns the utf-8 from google into python unicode

'loads' basically parses and turns the data from google into an object, exactly as we see in the first picture, so it retrieves a dictionary - if the status is not "OK" (which is part of the dictionary) then it fails

the first line 'lat = ' after that goes down the dictionary, results[0] because that's a list, then down to geometry, location then finally lat

The compute resources to run these APIs are not “free”

The data provided by these APIs is usually valuable

The data providers might limit the number of requests per day, demand an API “key”, or even charge for usage

They might change the rules as things progress...

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '5'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url)
    data = connection.read().decode()
    headers = dict(connection.getheaders())
    print('Remaining', headers['x-rate-limit-remaining'])
    js = json.loads(data)
    print(json.dumps(js, indent=4))

    for u in js['users']:
        print(u['screen_name'])
        s = u['status']['text']
        print('  ', s[:50])
```

twurl is to deal with the authentication part of the twitter api

url = … is again for authorization and security

data will decode the utf-8 and convert the json into a string

urllib doesn’t retrieve the headers, so getheaders will get them back, dict makes them into a dictionary

then next print line is twitter communicating how many more api requests you get per day (x-rate)

loads parses the data

dumps takes the json structure and makes it neater – as seen here below

you get an object that has users which is an array, and then each user within that is an object

so back to the code, u in js[‘users’] will loop through the users array, looking for screen name and the first 50 characters (text) in their status

```
Enter Twitter Account:drchuck
Retrieving https://api.twitter.com/1.1/friends ...
Remaining 14
{
    "users": [
        {
            "status": {
                "text": "@jazzychad I just bought one .__.",
                 "created_at": "Fri Sep 20 08:36:34 +0000 2013",
            },
            "location": "San Francisco, California",
            "screen_name": "leahculver",
            "name": "Leah Culver",
        },
        {
            "status": {
                "text": "RT @WSJ: Big employers like Google ...",
                 "created_at": "Sat Sep 28 19:36:37 +0000 2013",
            },
            "location": "Victoria Canada",
            "screen_name": "_valeriei",
            "name": "Valerie Irvine",
    ],
}
Leahculver
   @jazzychad I just bought one .__._
Valeriei
   RT @WSJ: Big employers like Google, AT&amp;amp;T are h
Ericbollens
   RT @lukew: sneak peek: my LONG take on the good &a
halherzog
  Learning Objects is 10. We had a cake with the LO,
```

```
import urllib
import oauth
import hidden

def augment(url, parameters) :
    secrets = hidden.oauth()
    consumer = oauth.OAuthConsumer(secrets['consumer_key'], secrets['consumer_secret'])
    token = oauth.OAuthToken(secrets['token_key'],secrets['token_secret'])
    oauth_request = oauth.OAuthRequest.from_consumer_and_token(consumer,
        token=token, http_method='GET', http_url=url, parameters=parameters)
    oauth_request.sign_request(oauth.OAuthSignatureMethod_HMAC_SHA1(), consumer, token)
    return oauth_request.to_url()


https://api.twitter.com/1.1/statuses/user_timeline.json?count=2
&oauth_version=1.0&oauth_token=101...SGI&screen_name=drchuck&oa
uth_nonce=09239679&oauth_timestamp=1380395644&oauth_signature=r
LK...BoD&oauth_consumer_key=h7Lu...GNg&oauth_signature_method=H
MAC-SHA1
```

import hidden basically asks you to sign the url off so twitter knows its you – they need 4 things, consumer key & secret, token key & secret

once you've given this exclusive data into hidden.py file, the purple part is the link, the import libraries do the green for you

```
import urllib.request, urllib.parse, urllib.error
import oauth
import hidden

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

def augment(url, parameters):
    secrets = hidden.oauth()
    consumer = oauth.OAuthConsumer(secrets['consumer_key'],
                                   secrets['consumer_secret'])
    token = oauth.OAuthToken(secrets['token_key'], secrets['token_secret'])

    oauth_request = oauth.OAuthRequest.from_consumer_and_token(consumer,
                    token=token, http_method='GET', http_url=url,
                    parameters=parameters)
    oauth_request.sign_request(oauth.OAuthSignatureMethod_HMAC_SHA1(),
                               consumer, token)
    return oauth_request.to_url()
```

this is the program written to define 'augment' – it uses OAuth, built into python, to sign the api using your 4 keys/secret

for the actual code below, this url with augment deals creates a url that deals with all the security

```
import urllib.request, urllib.parse, urllib.error
from twurl import augment
import ssl

# https://apps.twitter.com/
# Create App and get the four strings, put them in hidden.py

print('* Calling Twitter...')
url = augment('https://api.twitter.com/1.1/statuses/user_timeline.json',
              {'screen_name': 'drchuck', 'count': '2'})
print(url)

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

connection = urllib.request.urlopen(url, context=ctx)
data = connection.read()
print(data)
```

ssl certificates part is to deal with the https url as twitter doesn't have enough certificates, so this part turns them off

context=ctx turns that security off

after we get the headers also (screenshot cut off)

without keys/secrets you should get a 401 error, for authorization

```
https://api.twitter.com/1.1/statuses/user_timeline.json?oauth_consumer_key=ysf5Q
qtbSANXkHSpnagZhF2k5&oauth_timestamp=1483320693&oauth_nonce=54857909&oauth_versi
on=1.0&screen_name=drchuck&count=2&oauth_token=10185562-Lwn6Mg4QcErb4Mmm4HhJ3Xxr
0flBXgqGB2m4fgH8j&oauth_signature_method=HMAC-SHA1&oauth_signature=GHFmS9LNuC6Vv
cl89zfp7PU7opM%3D
```

timestamps/nonce basically means you can't replay the url even if you see all keys – so seeing all of this still means you cannot break into my account

also doesn't show the secrets so it is not a security breach

```
2379591071:code3 csev$ python3 twtest.py
* Calling Twitter...
https://api.twitter.com/1.1/statuses/user_timeline.json?oauth_consumer_key=ysf5Q
qtbSANXkHSpnagZhF2k5&oauth_timestamp=1483320693&oauth_nonce=54857909&oauth_versi
on=1.0&screen_name=drchuck&count=2&oauth_token=10185562-Lwn6Mg4QcErb4Mmm4HhJ3Xxr
0flBXgqGB2m4fgH8j&oauth_signature_method=HMAC-SHA1&oauth_signature=GHFmS9LNuC6Vv
cl89zfp7PU7opM%3D
b'[{"created_at":"Mon Jan 02 00:38:55 +0000 2017","id":815718984347750400,"id_st
r":"815718984347750400","text":"Web Services in Python 3 - https:\\/\\/t.co\\/Oo
JYpgJyjl https:\\/\\/t.co\\/spSrJuiOOb watch people code products live https:\\/
\\/t.co\\/LfNshs1F5G","truncated":false,"entities":{"hashtags":[],"symbols":[],"
user_mentions":[],"urls":[{"url":"https:\\/\\/t.co\\/OoJYpgJyjl","expanded_url":
"http:\\/\\/py4e.com","display_url":"py4e.com","indices":[27,50]},{"url":"https:
\\/\\/t.co\\/spSrJuiOOb","expanded_url":"http:\\/\\/livecoding.tv","display_url"
:"livecoding.tv","indices":[51,74]},{"url":"https:\\/\\/t.co\\/LfNshs1F5G","expa
nded_url":"https:\\/\\/www.livecoding.tv\\/drchuck\\/","display_url":"livecoding
.tv\\/drchuck\\/","indices":[107,130]}]},"source":"\\u003ca href=\\"http:\\/\\/t
witter.com\\" rel=\\"nofollow\\"\\u003eTwitter Web Client\\u003c\\/a\\u003e","in
_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id"
:null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id"
:10185562,"id_str":"10185562","name":"drchuck","screen_name":"drchuck","location
":"iPhone: 37.868347,-122.255997","description":"I am on the faculty @UMSI, I wo
rk on the open source @sakaiproject, and I teach free online Python courses on C
oursera  https:\\/\\/t.co\\/T3NbCOz8FR","url":"http:\\/\\/t.co\\/5szJLzBM4S","en
```

basically because there was no decode, you get a byte array, a raw set of bytes that came from internet in utf-8

```
{'cache-control': 'no-cache, no-store, must-revalidate, pre-check=0, post-check=
0', 'connection': 'close', 'content-disposition': 'attachment; filename=json.jso
n', 'content-length': '5939', 'content-type': 'application/json;charset=utf-8',
'date': 'Mon, 02 Jan 2017 01:33:11 GMT', 'expires': 'Tue, 31 Mar 1981 05:00:00 G
MT', 'last-modified': 'Mon, 02 Jan 2017 01:33:11 GMT', 'pragma': 'no-cache', 'se
rver': 'tsa_b', 'set-cookie': 'guest_id=v1%3A148332079143445525; Domain=.twitter
.com; Path=/; Expires=Wed, 02-Jan-2019 01:33:11 UTC', 'status': '200 OK', 'stric
t-transport-security': 'max-age=631138519', 'x-access-level': 'read-write', 'x-c
onnection-hash': '2660ca7a4ad5382dd47b00756fd94ecb', 'x-content-type-options': '
nosniff', 'x-frame-options': 'SAMEORIGIN', 'x-rate-limit-limit': '900', 'x-rate-
limit-remaining': '898', 'x-rate-limit-reset': '1483321594', 'x-response-time':
'66', 'x-transaction': '0013650800a89825', 'x-twitter-response-tags': 'BouncerCo
mpliant', 'x-xss-protection': '1; mode=block'}
```

these are the headers

take note of the x-rate-limit-remaining number, toward the bottom – twitter is tracking how many times you use the request

```
while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '2'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()
    print(data[:250])
    headers = dict(connection.getheaders())
    # print headers
    print('Remaining', headers['x-rate-limit-remaining'])
```

similar as before except we decode the utf-8 json into Unicode string, print the first 250 characters, retrieve the headers and print the remaining number of url requests

```
while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '5'})
    print('Retrieving', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()
    headers = dict(connection.getheaders())
    print('Remaining', headers['x-rate-limit-remaining'])
    js = json.loads(data)
    print(json.dumps(js, indent=4))

    for u in js['users']:
        print(u['screen_name'])
        s = u['status']['text']
        print('  ', s[:50])
```

same as before except now we add loads and dumps, which parses and makes it neater

js is a dictionary but when we do js['users'] that's a list

print the first 50 characters of the status

do that for first 5 friends due to count being 5

first you get json debug code but after you get what you want

```
for u in js['users']:
    print(u['screen_name'])
    if 'status' not in u:
        print('  Not status found')
        continue
    s = u['status']['text']
    print('  ', s[:50])
```

for example if the status is bringing back an error, or cannot be found, can add this section to the code

```
import urllib.request, urllib.parse, urllib.error
import json

url = 'http://py4e-data.dr-chuck.net/comments_1307703.json'
uh = urllib.request.urlopen(url)
data = uh.read().decode()

js = json.loads(data)
sum = 0

for c in js['comments']:
    num = int(c['count'])
    sum = sum + num

print(sum)
```


```
{
"note":"This file contains the sample data for testing",
"comments":[
  {
    "name":"Romina",
    "count":97
  },
  {
    "name":"Laurie",
    "count":97
  },
  {
    "name":"Bayli",
    "count":90
  },
  {
    "name":"Siyona",
    "count":90
  },
  {
    "name":"Taisha",
    "count":88
  },
  {
    "name":"Alanda",
    "count":87
  },
```


read url, open it, decode the data and store it

js parses it

set count to 0

read through each item in dictionary

retrieve the count for each comment

sum and print

example of the json data from url

```
import urllib.request, urllib.parse, urllib.error
import json
import ssl

api_key = False
# If you have a Google Places API key, enter it here
# api_key = 'AIzaSy___IDByT70'
# https://developers.google.com/maps/documentation/geocoding/intro

if api_key is False:
    api_key = 42
    serviceurl = 'http://py4e-data.dr-chuck.net/json?'
else :
    serviceurl = 'https://maps.googleapis.com/maps/api/geocode/json?'

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    address = input('Enter location: ')
    if len(address) < 1: break

    parms = dict()
    parms['address'] = address
    if api_key is not False: parms['key'] = api_key
    url = serviceurl + urllib.parse.urlencode(parms)

    print('Retrieving', url)
    uh = urllib.request.urlopen(url, context=ctx)
    data = uh.read().decode()
    print('Retrieved', len(data), 'characters')

    try:
        js = json.loads(data)
    except:
        js = None

    if not js or 'status' not in js or js['status'] != 'OK':
        print('==== Failure To Retrieve ====')
        print(data)
        continue

    id = js['results'][0]['place_id']
    print(id)
```

program will prompt for a location, contact a web service and retrieve json for the web service and parse that data

retrieves the first place_id from the json (a place id is a textual identifier that uniquely identifies a place as within google maps)

http://py4e-data.dr-chuck.net/json?

this api uses the same parameter (address) as the google api

this api also has no rate limit so you can test as often as you like

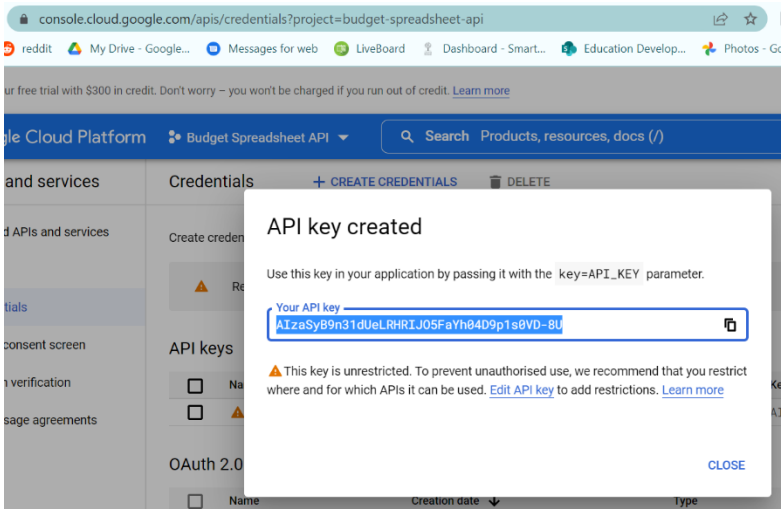
if you visit the url with no parameters, you get "No address..." response.

to call the api, you need to include a key= parameter and provide the address that you are requesting as the address= parameter that is properly URL encoded using the urllib.parse.urlencode() function

Credential types

The credentials required depends on the type of data, platform, and access methodology of your app. Below are the available credential types:

- **API key** – Use this credential to access publicly-available data anonymously in your app.
- **OAuth client ID** – Use this credential to authenticate as an end user and access their data. Requires your app to request and receive consent from the user.
- **Service account** – Use this credential to authenticate as a robot service account or to access resources on behalf of Google Workspace or Cloud Identity users through domain-wide delegation.



gmail API created through google cloud

AIzaSyB9n31dUeLRHRIJO5FaYh04D9p1s0VD-8U

OAuth client ID credentials

To authenticate as an end user and access user data in your app, you need to create one or more *OAuth 2.0 Client IDs*. A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, you need to create a separate client ID for each platform.

Create credentials for a service account

You must obtain credentials in the form of a public/private key pair. These credentials are used by your code to authorize service account actions within your app.

To obtain credentials for your service account:

1. Open the Google Cloud Console.
2. At the top-left, click Menu ☰ > **IAM & Admin** > **Service Accounts**.
3. Select your service account.
4. Click **Keys** > **Add keys** > **Create new key**.
5. Select **JSON**, then click **Create**.

   Your new public/private key pair is generated and downloaded to your machine as a new file. This file is the only copy of this key. For information about how to store your key securely, see Managing service account keys.
6. Click **Close**.

2.16: Object Oriented

Program - input > process > output

Objects - smaller blocks of code within a program needed for the process – it is self-contained code and data

Class – a template (a blueprint, like a cookie cutter)

Method/message – a defined capability of a class (part of class and object, different functions within a class (but also object), cookie cutter shapes)

Field/attribute – a bit of data in a class

Object/instance – a particular instance of a class (the actual thing, the cookie)

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
```

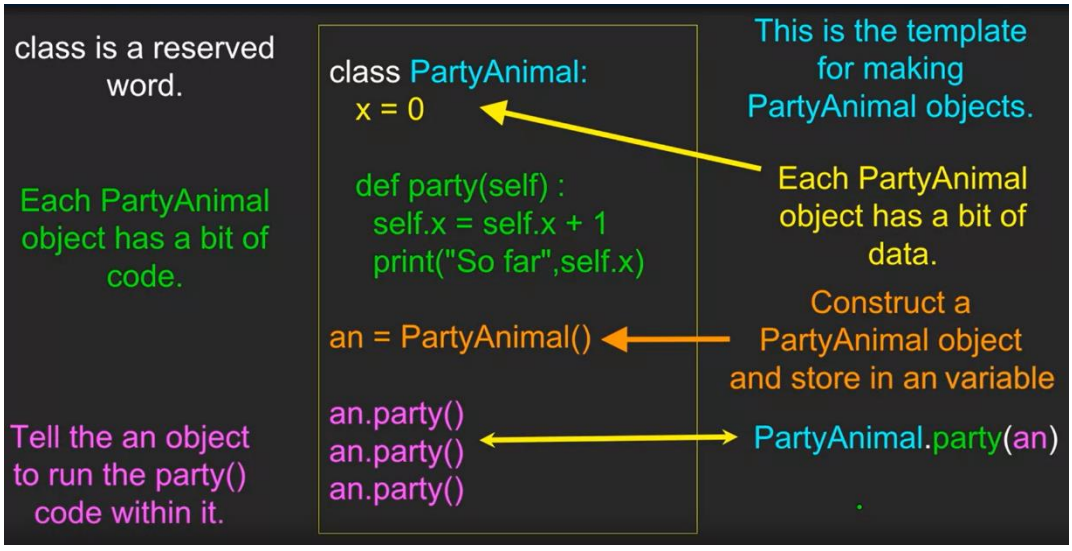
x is an object of type class, the class is string

instance of a string class

```
>>> dir(y)
[… 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

these are the methods for lists

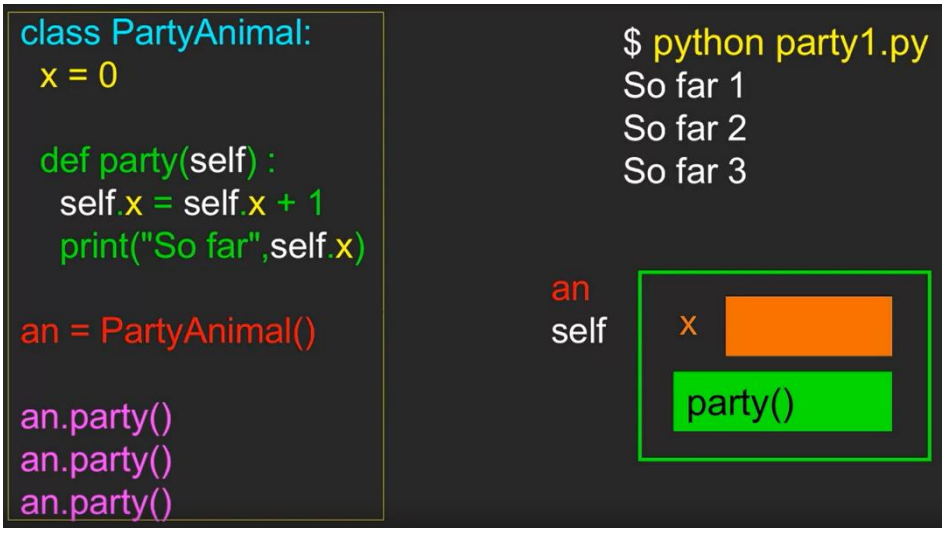
there are different methods for strings, floats etc



name of class: Party Animal

method inside class

so these last lines are basically calling the party animal class, and the party method within it, and passing an as the argument – so in the party method it wouldn't be self, it would be an (i.e. an.x = an.x + 1 etc etc)



to explain the code above, look at the output:

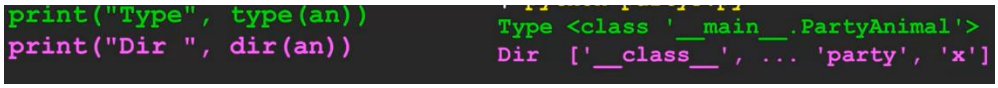
so the class and method is just a template, nothing happens

we create our 'an' variable, an instance of a class (and therefore will make an object in 'an')

so with 'an' created, it runs through the class and defines things, i.e. x = 0 and party() – so now 'an' points to this object, x=0 and party()

now the last three lines – the first one, it goes up and calls the party method, except we already have 'self' there where instead we are trying to call 'an' – so self will now also point to the same object temporarily (x=0, party())

its temporary as once it runs through the party() method, x will become 1, therefore printing this out, and storing 1 in 'x' – this just repeats 3 times because there are 3 lines



dir tells you whats inside (in this case inside the class)

```
class Employee:
    pass
```

empty class (without pass it would give an error)

there are no functions here (functions linked to a class are called methods) and there are no attributes

```
emp_1 = Employee()
emp_2 = Employee()
```

these variables are unique instances of the class employee

can manually add instance variables, called attributes

but takes long to add each attribute to each class

```
def __init__(self, first, last, pay):
    self.first = first
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@company.com'
```

to automate this info for all employees, create an init method

pass the self argument through (it is convention)

methods within a class receive the instance as the first argument automatically

self.first etc are all instance variables – self.first=first, they don't have to be the same (it can be self.fname=first)

so the init method takes the instance (self) and the first, last name and pay as arguments – because the instance is passed automatically, when we have our class variables (e.g. emp_1) we do not need to pass self

```
emp_1 = Employee('Corey', 'Schafer', 50000)
emp_2 = Employee('Test', 'User', 60000)
```

should look like this now

employee 1 is passed in as self, then will set all the attributes (so the self.first = first will become, emp_1.first = Corey)

```
print('{} {}'.format(emp_1.first, emp_1.last))
```

lets say I wanted to get employees full names, I can manually write this

```
def fullname(self):
    return '{} {}'.format(self.first, self.last)
```

each method in a class automatically takes the instance of the first argument

so define a new method, only self as the argument as we have already defined the other arguments before, those attributes already exist

just replace emp_1 for self, so it works for all employees

```
print(emp_1.fullname())
```

to get the result

the brackets are because this fullname is a method, not an attribute

```
emp_1.fullname()
Employee.fullname(emp_1)
```

we can use the class name – these two lines do the same thing

we have to manually pass the instance as an argument when using the class

of course when using emp_1 we have already told the method what instance (emp_1) where as when we use the class Employee it doesn't know, so we must pass emp_1 in the argument

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('I am constructed')

   def party(self) :
     self.x = self.x + 1
     print('So far',self.x)

   def __del__(self):
     print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

```
$ python party4.py
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

init is a constructor, del is a destructor

to understand this better I broke the code up:

```
class PartyAnimal:
    x = 0

    def __init__(self):
        print('I am consutrcted')

    def party(self) :
        self.x = self.x + 1
        print('So far', self.x)

    def __del__(self):
        print('I am destructed', self.x)

an = PartyAnimal()
```

```
I am consutrcted
I am destructed 0
```

```
class PartyAnimal:
    x = 0

    def __init__(self):
        print('I am consutrcted')

    def party(self) :
        self.x = self.x + 1
        print('So far', self.x)

    def __del__(self):
        print('I am destructed', self.x)

an = PartyAnimal()
an.party()
```

```
I am consutrcted
So far 1
I am destructed 1
```

```
class PartyAnimal:
    x = 0

    def __init__(self):
        print('I am consutrcted')

    def party(self) :
        self.x = self.x + 1
        print('So far', self.x)

    def __del__(self):
        print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
```

```
I am consutrcted
So far 1
So far 2
I am destructed 2
```

(i) Constructed the class, automatically destructs the class

(ii) Constructs the class, calls the party method once, hence x = 1, therefore destructs after 1 loop

(iii) Calls the method twice, after first one x = 1, but after the second, x becomes 2, destructs after 2 loops

(iv) (original code above these three): after the loops set an = 42 so no longer points to the class object, but instead only at 42

Storing the PartyAnimal class inside variable ‘an’ is not enough to run through the method within the class – it needs to be called on separately using the object stored in the variable (i.e. an.party())

We can create lots of objects - the class is the template for the object

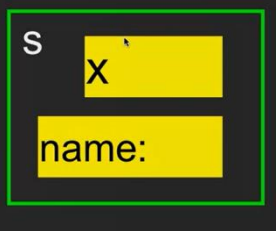
We can store each distinct object in its own variable

We call this having multiple instances of the same class

Each instance has its own copy of the instance variables

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, z):
     self.name = z
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

We have two independent instances.



idea of this code is to show how x can be 1 and 2

first pass in an argument to the constructor (z) – this will be a placeholder for a name

create an object in variable ‘s’ for a PartyAnimal class with the name Sally passed in:
so in this object ‘s’, it contains two bits of information, x = 0 and name = Sally
then call the party method using this information, so x becomes 1

do the same thing but create a new instance of the class PartyAnimal, this time with name of Jim:
in j the same info to begin with, and when the party method is called x is also 1

Last thing we call party method but this time for s again, x will become 2

‘Subclasses’ are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

inherit is when you can copy classes and their attributes but also change and add things

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan is a class which extends PartyAnimal. It has all the capabilities of PartyAnimal and more.

FootballFan class is an extension, it just adds on the code to the existing PartyAnimal class

‘s’ points to x = 0, name is Sally, after the party method is called x = 1

‘j’ however points to x = 0, name is Jim, after the party method x = 1, but also after touchdown method is called, points = 7

Class - a template

Attribute – A variable within a class

Method - A function within a class

Object - A particular instance of a class

Constructor – Code that runs when an object is created

Inheritance - The ability to extend a class to make a new class.

definitions

Chapter 3: Pandas guide

3.1: Pandas Basics

```
In [2]: titanic_df = pd.read_excel('titanic3.xls','titanic3', index_col=None, na_values=['NA']) #read excel sheet to dataframe
In [3]: titanic_df.head()
Out[3]:
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2														Montreal,

after importing (import pandas as pd):

pd.read gets the file

.head will create a table

```
titanic_df.describe()
```

	pclass	survived	age	sibsp	parch	fare	body
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	121.000000
mean	2.294882	0.381971	29.881135	0.498854	0.385027	33.295479	160.809917

describe tells us the count, mean, the stats

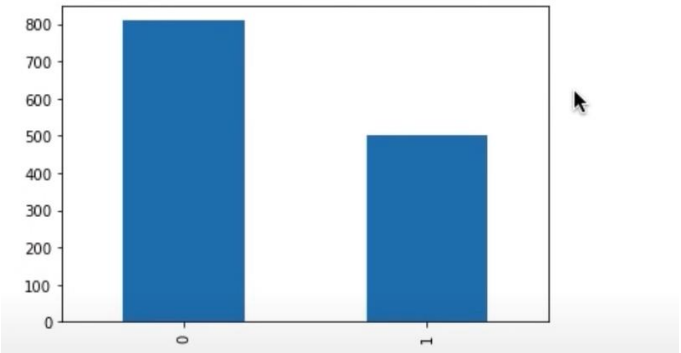
```
titanic_df.drop(['ticket','cabin','boat','body'],axis=1).head()
```

	pclass	survived	name	sex	age	sibsp	parch	fare	embarked	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	211.3375	S	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	151.5500	S	Montreal, PQ / Chesterville, ON
2										Montreal, PQ / Chesterville,

drop deletes some columns

head makes the table

```
pd.value_counts(titanic_df['survived']).plot.bar()
<matplotlib.axes._subplots.AxesSubplot at 0x10d776630>
```



take the counts of only survived column and make a bar chart

```
titanic_df['survived'].mean()
0.3819709702062643
```

mean of survived

```
titanic_df.groupby(['sex']).mean()
```

mean but grouped by male or female

	pclass	survived	age	sibsp	parch	fare	body
sex							
female	2.154506	0.727468	28.687071	0.652361	0.633047	46.198097	166.62500
male	2.372479	0.190985	30.585233	0.413998	0.247924	26.154601	160.39823

```
titanic_df.groupby(['sex','pclass']).mean()
```

		survived	age	sibsp	parch	fare	body
sex	**pclass**						
female	**1**	0.965278	37.037594	0.555556	0.472222	109.412385	NaN
	2	0.886792	27.499191	0.500000	0.650943	23.234827	52.000000
	3	0.490741	22.185307	0.791667	0.731481	15.324250	183.000000
male	**1**	0.340782	41.029250	0.340782	0.279330	69.888385	162.828571
	2	0.146199	30.815401	0.327485	0.192982	19.904946	171.233333
	3	0.152130	25.962273	0.470588	0.255578	12.415462	151.854167

group by sex and then class

find the mean

```
titanic_df[titanic_df['age']<18].groupby(['sex','pclass']).mean()
```

same thing but for under 18's

		survived	age	sibsp	parch	fare	body
sex	**pclass**						
female	**1**	0.875000	14.125000	0.500000	0.875000	104.083337	NaN
	2	1.000000	8.273150	0.666667	1.166667	27.998844	NaN
	3	0.543478	8.416667	1.456522	1.043478	18.284148	328.0
male	**1**	0.857143	9.845243	0.571429	1.714286	129.752371	NaN
	2	0.733333	6.222220	0.600000	0.933333	31.750280	NaN
	3	0.233333	9.838888	1.966667	1.016667	21.677570	65.5

3.2: Loading Data

```
import pandas as pd

df = pd.read_csv('pokemon_data.csv')

print(df)
```

```
      #                    Name    Type 1  Type 2   HP  Attack  Defense
0     1                Bulbasaur     Grass  Poison   45      49       49
1     2                  Ivysaur     Grass  Poison   60      62       63
2     3                 Venusaur     Grass  Poison   80      82       83
3     3      VenusaurMega Venusaur     Grass  Poison   80     100      123
4     4               Charmander      Fire     NaN   39      52       43
```

how to load data (csv) in to a dataframe:

```
df_xlsx = pd.read_excel('pokemon_data.xlsx')
print(df_xlsx.head(3))
```

Doesn't have to be csv files, can be xlsx (excel) files too

```
df = pd.read_csv('pokemon_data.txt', delimiter='\t')

print(df)
```

this is so you can read in txt files – you have to put the delimeter as the columns in the txt files are separated by tabs, not commas

3.3: Reading Data

```
## Read Headers
print(df.columns)

## Read each Column

## Read Each Row

## Read a specific location (R,C)
```

```
Index(['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
```

```
print(df.head(3))
```

```
   #       Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed
0  1  Bulbasaur  Grass  Poison  45      49       49       65       65     45
1  2    Ivysaur  Grass  Poison  60      62       63       80       80     60
2  3   Venusaur  Grass  Poison  80      82       83      100      100     80
```

only prints first three rows

```
print(df.tail(3))
```

```
       #                 Name   Type 1 Type 2  HP  Attack  Defense  Sp. Atk
797  720  HoopaHoopa Confined  Psychic  Ghost  80     110       60      150
798  720   HoopaHoopa Unbound  Psychic   Dark  80     160       60      170
799  721            Volcanion     Fire  Water  80     110      120      130

     Sp. Def  Speed  Generation  Legendary
797      130     70           6       True
798      130     80           6       True
799       90     70           6       True
```

bottom three rows

```
## Read each Column
print(df['Name'])

## Read Each Row

## Read a specific location (R,C)
```

```
0                       Bulbasaur
1                         Ivysaur
2                        Venusaur
3                VenusaurMega Venusaur
```

```
print(df['Name'][0:5])
```

first 5 rows in that column

```
print(df.Name)
```

can also do this

```
print(df[['Name', 'Type 1', 'HP']])
```

different columns

```
## Read Each Row
print(df.iloc[1])
```

for row 1 (so second row)

iloc means integer location

```
#                2
Name           Ivysaur
Type 1           Grass
Type 2          Poison
HP                  60
Attack              62
Defense             63
Sp. Atk             80
Sp. Def             80
Speed               60
Generation           1
Legendary        False
Name: 1, dtype: object
```

```
print(df.iloc[1:4])
```

```
## Read a specific location (R,C)
print(df.iloc[2,1])
```

third row, second column

3.4: Iterations

```
for index, row in df.iterrows():
    print(index, row['Name'])

## Read a specific location (R,C)
#print(df.iloc[2,1])
```

```
32 Sandshrew
33 Sandslash
34 Nidoran (Female)
35 Nidorina
36 Nidoqueen
37 Nidoran (Male)
38 Nidorino
39 Nidoking
40 Clefairy
41 Clefable
```

so this iterates through each row and prints out the index and the rows (this time we specified names only) – see how it prints out like a list

3.5: Conditions/Sorting

```
df.loc[df['Type 1'] == "Fire"]

## Read a specific location (R,C)
#print(df.iloc[2,1])
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def
4	4	Charmander	Fire	NaN	39	52	43	60	50
5	5	Charmeleon	Fire	NaN	58	64	58	80	65
6	6	Charizard	Fire	Flying	78	84	78	109	85
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85

loc functions finds data without use of index

can see we only searched for fire pokemon

```
df.describe()
```

	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
count	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.00000
mean	362.813750	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	3.32375
std	208.343798	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	1.66129
min	1.000000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	1.00000

again, describe shows the stats

```
df.sort_values('Name')
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk
510	460	Abomasnow	Grass	Ice	90	92	75	92
511	460	AbomasnowMega Abomasnow	Grass	Ice	90	132	105	132
68	63	Abra	Psychic	NaN	25	20	15	105
392	359	Absol	Dark	NaN	65	130	60	75
393	359	AbsolMega Absol	Dark	NaN	65	150	60	115

sort column Name, by alphabetical order

```
df.sort_values('Name', ascending=False)
```

	#	Name	Type 1	T
794	718	Zygarde50% Forme	Dragon	G
695	634	Zweilous	Dark	D

reverses the order

```
df.sort_values(['Type 1', 'HP'], ascending=[1,0])
```

	#	Name	Type 1	Type 2	HP	Attack	De
520	469	Yanmega	Bug	Flying	86	76	
698	637	Volcarona	Bug	Fire	85	60	
231	214	Heracross	Bug	Fighting	80	125	
232	214	HeracrossMega Heracross	Bug	Fighting	80	185	

sorting two columns, but first one is ascending, second one descending

3.6: Changing the Dataframe

```
df['Total'] = df['HP'] + df['Attack'] + df['Defense'] + df['Sp. Atk'] + df['Sp. Def'] + df['Speed']

df.head(5)
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False	318
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False	405
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False	525

added a new column, Total

```
df = df.drop(columns=['Total'])
```

to remove column, rewrite dataframe and drop the column

```
df['Total'] = df.iloc[:, 4:10].sum(axis=1)

df.head(5)
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False	273
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False	345

[:, 4:9] means all rows, 4th-8th column

axis=1 means add horizontally, not vertically (that would be axis=0)

```
cols = list(df.columns)
df = df[cols[0:4] + [cols[-1]]+cols[4:12]]

df.head(5)
```

	#	Name	Type 1	Type 2	Legendary	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
0	1	Bulbasaur	Grass	Poison	False	864	45	49	49	65			

moving total column in the middle

3.7: Saving the Data

```
df.to_csv('modified.csv')
```

saves file to csv under the name modified

removes index column

```
df.to_csv('modified.csv', index=False)
```

can do the same for excel (.to_excel)

```
df.to_csv('modified.txt', index=False, sep='\t')
```

same for text files - .to.csv(modified.txt) separated by tabs

3.8: Filtering Data

```
df.loc[df['Type 1'] == 'Grass']
```

	#	Name	Type 1	Type 2	Total	HP	Attack
0	1	Bulbasaur	Grass	Poison	318	45	49
1	2	Ivysaur	Grass	Poison	405	60	62
2	3	Venusaur	Grass	Poison	525	80	82

filters out all data that isn't type 1 = grass

```
df.loc[(df['Type 1'] == 'Grass') & (df['Type 2'] == 'Poison')]
```

```
df.loc[(df['Type 1'] == 'Grass') | (df['Type 2'] == 'Poison')]
                & (df['HP'] > 70)]
```

using or operator

```
df.loc[df['Name'].str.contains('Mega')]
```

	#	Name	Type 1	Type
3	3	VenusaurMega Venusaur	Grass	Pois
7	6	CharizardMega Charizard X	Fire	Drag
8	6	CharizardMega Charizard Y	Fire	Flyi
12	9	BlastoiseMega Blastoise	Water	Na
19	15	BeedrillMega Beedrill	Bug	Pois
23	18	PidgeotMega Pidgeot	Normal	Flyi

find all rows with 'Mega' in the name

```
df.loc[~df['Name'].str.contains('Mega')]
```

this does the opposite ^
finds all strings without that word Mega

3.9: Resetting Index

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120
50	45	Vileplume	Grass	Poison	490	75	80	85	110	90

can see the index is from the old dataframe

```
new_df = new_df.reset_index()

new_df
```

reset index gives a new column but also keeps the old one

	index	#	Name
0	2	3	Venusaur
1	3	3	VenusaurMega Venusaur
2	50	45	Vileplume
3	77	71	Victreebel
4	652	591	Amoonguss

drop=True, inplace=True removes the old index

```
new_df.reset_index(drop=True, inplace=True)

new_df
```

	#	Name	Type 1	Type 2	Total
0	3	Venusaur	Grass	Poison	525
1	3	VenusaurMega Venusaur	Grass	Poison	625
2	45	Vileplume	Grass	Poison	490
3	71	Victreebel	Grass	Poison	490
4	591	Amoonguss	Grass	Poison	464

3.10: Regex (Regular Expression) Filtering

```
import re

df.loc[df['Type 1'].str.contains('Fire|Grass', regex=True)]
```

string contains is case sensitive

using regular expressions

| means or

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65

```
df.loc[df['Type 1'].str.contains('fire|grass', flags=re.I, regex=True)]
```

this ignores the case

```
df.loc[df['Name'].str.contains('pi[a-z]*', flags=re.I, regex=True)]
```

using regex we can find parts of a string that contain 'pi' for example

[a-z]* means any letter 0 or more times

put ^pi to mean START with pi

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense
13	10	Caterpie	Bug	NaN	195	45	30	35
20	16	Pidgey	Normal	Flying	251	40	45	40
21	17	Pidgeotto	Normal	Flying	349	63	60	55
22	18	Pidgeot	Normal	Flying	479	83	80	75
23	18	PidgeotMega Pidgeot	Normal	Flying	579	83	80	80

3.11: Conditional Changes

```
df.loc[df['Type 1'] == 'Fire', 'Type 1'] = 'Flamer'

df
```

change all fire types to flamer

```
df.loc[df['Type 1'] == 'Fire', 'Legendary'] = True
```

important - if the type 1 is fire, it has to be legendary

```
df.loc[df['Total'] > 500, ['Generation','Legendary']] = 'TEST VALUE'

df
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	TEST VALUE	TEST VALUE
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	TEST VALUE	TEST VALUE

if the total is above 500, make the generation and legendary column 'TEST VALUE' for those pokemon

```
df.loc[df['Total'] > 500, ['Generation','Legendary']] = ['Test 1', 'Test 2']
```

can change each condition separately

3.12: Aggregate Statistics (GroupBy)

```
df.groupby(['Type 1']).mean()
```

the mean of all the columns for each type

Type 1	#	Total	HP	Attack	Defense	Sp. Atk	Sp. Def
Bug	334.492754	378.927536	56.884058	70.971014	70.724638	53.869565	64.797101
Dark	461.354839	445.741935	66.806452	88.387097	70.225806	74.645161	69.516129

```
df.groupby(['Type 1']).mean().sort_values('Defense', ascending=False)
```

this sorts the means by defence – and reverses the order so the highest comes first

```
df.groupby(['Type 1']).sum()
```

Type 1	#	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Bug	23080	26146	3925	4897	4880	3717	4471	4256
Dark	14302	13818	2071	2740	2177	2314	2155	2361
Dragon	15180	17617	2666	3588	2764	3099	2843	2657
Electric	15994	19510	2631	3040	2917	3961	3243	3718

you can sum all the columns belonging to each type

```
df.groupby(['Type 1']).count()
```

Type 1	#	Name	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Bug	69	69	52	69	69	69	69	69	69	69
Dark	31	31	21	31	31	31	31	31	31	31
Dragon	32	32	21	32	32	32	32	32	32	32
Electric	44	44	17	44	44	44	44	44	44	44
Fairy	17	17	2	17	17	17	17	17	17	17

or just the frequency (count)

```
df['count'] = 1
```

adds a column to dataframe and sets all rows to 1

```
df.groupby(['Type 1']).count()
```

counts the type 1 column and adds to the dataframe

```
2 df.groupby(['pclass', 'sex'])['survived'].sum()
```

```
pclass  sex
1       female    139
        male       61
2       female     94
        male       25
3       female    106
        male       75
Name: survived, dtype: int64
```

returns a multi-levelled index table

```
1 # Groupby - Multiple Groups AS Columns
2 df.groupby(['pclass', 'sex'], as_index=False)['survived'].sum()
```

	pclass	sex	survived
0	1	female	139
1	1	male	61
2	2	female	94
3	2	male	25
4	3	female	106
5	3	male	75

this is to remove the multi-level index

```
1 # Multiple Functions
2 df.groupby(['pclass'])['survived'].agg(['mean','sum'])
```

pclass	mean	sum
1	0.619195	200
2	0.429603	119
3	0.255289	181

group by class, data returned is to be from survived column, finding the mean and sum of said column

```
1 df['age_bins'] = pd.cut(df['age'], bins=3, labels=('young', 'middle_age', 'old'))
2 df
```

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	age_bins
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO	middle_age
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON	young
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	young
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON	middle_age

new column – the cut function splits the new column into 3 equal parts (bins), then we label them

3.13: Working with Large Datasets

```
for df in pd.read_csv('modified.csv', chunksize=5):
    print("CHUNK DF")
    print(df)
```

chunksize 5 means read in 5 rows at a time

```
CHUNK DF
     #      Name  Type 1  Type 2  Total  HP  Attack  Defense  Sp. Atk
25  20  Raticate  Normal     NaN    413  55      81       60       50
26  21   Spearow  Normal  Flying    262  40      60       30       31
27  22    Fearow  Normal  Flying    442  65      90       65       61
28  23     Ekans  Poison     NaN    288  35      60       44       40
29  24     Arbok  Poison     NaN    438  60      85       69       65
```

first print statement is just to show that after every 5 rows it resets and does the loop again

```
new_df = pd.DataFrame(columns=df.columns)

for df in pd.read_csv('modified.csv', chunksize=5):
    results = df.groupby(['Type 1']).count()

    new_df = pd.concat([new_df, results])
```

this creates a new dataframe but using the same columns as the original dataframe

it reads in data, 5 rows at a time

it creates a variable results that is based off the data being read in, and we look at the count of type 1

then using this data we load it into the new df as it is being read in

3.14: Index Max

```
np.random.seed(seed=42)

df = pd.DataFrame(data=np.random.randint(0, 100, (4,3)),
            columns=['Test1', 'Test2', 'Test3'],
             index=['Bob','Sally', 'Frank', 'Patty']
                  )
df
```

creates a dataframe with test scores but it is randomized

	Test1	Test2	Test3
Bob	51	92	14
Sally	71	60	20
Frank	82	86	74
Patty	74	87	99

```
df.idxmax(axis=1)
```

```
Bob      Test2
Sally    Test1
Frank    Test2
Patty    Test3
dtype: object
```

idxmax axis=1 finds the highest value in the row and returns the corresponding column header

finding which row's column had the highest value

```
df.idxmax(axis=0)
```

```
Test1     Frank
Test2       Bob
Test3     Patty
dtype: object
```

flip the axis

df = df.loc[df.groupby('company_name')['price'].idxmax()][['company_name', 'date']]

company_name	price	date
Google	101.2	12-02-22
Mirosoft	98.4	13-02-22
Microsoft	100.4	12-02-22
Google	101.4	13-02-22

this code groups by the company name, finds the max price and returns the company name & date

Chapter 4: SQL

4.1: Databases

Database - contains many tables

Relation (or table) - contains tuples and attributes

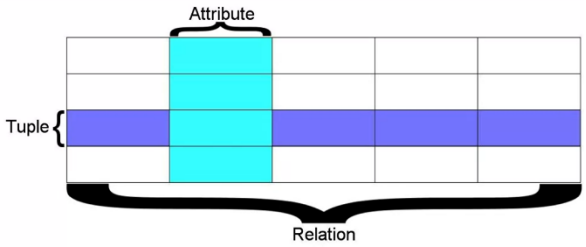
Tuple (or row) - a set of fields that generally represents an "object" like a person or a music track

Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row

table

row

column



relation defined as set of tuples that have the same attributes

tuple – represents an object & information

attribute – in one domain & conform to constraints

Columns / Attributes

TITLE	RATING	LEN	Rows / Tuples
About to Rock	3	354	
Who Made Who	4	252	
	Tables / Relations		

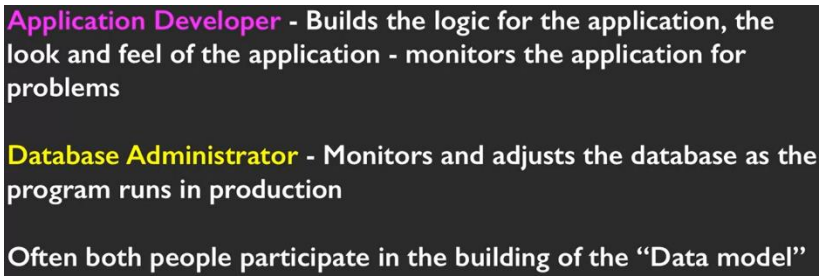
all the row headings are metadata about the database - called the schema

Structured Query Language is the language we use to issue commands to the database

create table, retrieve data, insert/update then delete data (CRUD)

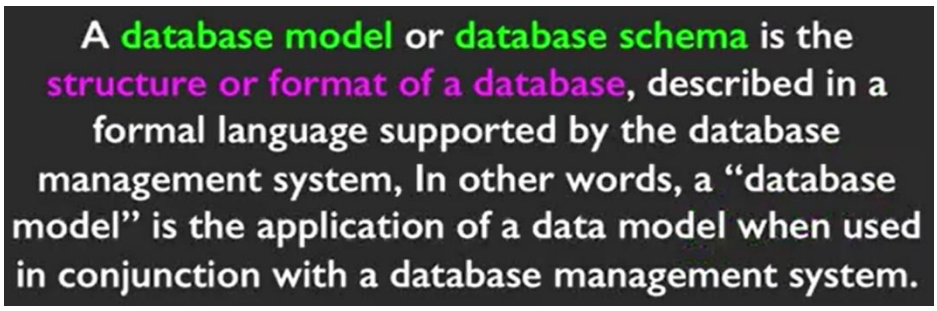
- speaking to databases used to take time but with abstraction there is a layer between the application (user) and the database
- this layer, we can say is an API, and it speeds the communication with the database because we can hide the complexity in it

- both data and software that understands the shape of the data is just a service, service orientated architecture
- we communicate through Oracle, using SQL (Structured Query Language) – universal language



AD writes code for application software which talks to database through SQL

DA talks to database directly using database tools and SQL



4.2: SQL Basics

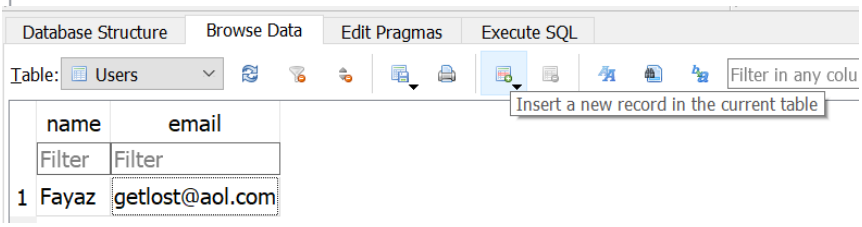
```
CREATE TABLE Users(
   name VARCHAR(128),
   email VARCHAR(128)
)
```

after creating a new database in SQLite:

create table is a keyword, the table name is 'Users'
create column 'name', which will contain variable characters (up to 128 characters)

this is our schema (contract)

when executed, browsing the data shows this table with headers name and email



putting records in to the table is done like this

```
INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
```

inserts a row into a table

```
DELETE FROM Users WHERE email='ted@umich.edu'
```

deleting without where clause will delete all rows

```
UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
```

updates name to Charles

```
SELECT * FROM Users
```

selects all columns (select basically is what we want to see)

```
SELECT COUNT(*) FROM Users
```

counts rows

```
SELECT * FROM Users WHERE email='csev@umich.edu'
```

selects from one column

```
SELECT * FROM Users ORDER BY email

SELECT * FROM Users ORDER BY name
```

ordering

```
import sqlite3

conn = sqlite3.connect('emaildb.sqlite')
cur = conn.cursor()
```

first import the library

two-step to open databases, one is connect, two is create a cursor – similar to handle but we pass SQL through the cursor

```
cur.execute('DROP TABLE IF EXISTS Counts')

cur.execute('''
CREATE TABLE Counts (email TEXT, count INTEGER)''')
```

you can use triple quotes, doesn't matter

you can connect to files that don't exist (emaildb.sqlite doesn't exist) – hence why our command has 'IF EXISTS', which it doesn't so that line does nothing

next it creates a table using a dictionary to store the columns

```
fname = input('Enter file name: ')
if (len(fname) < 1): fname = 'mbox-short.txt'
fh = open(fname)
for line in fh:
    if not line.startswith('From: '): continue
    pieces = line.split()
    email = pieces[1]
    cur.execute('SELECT count FROM Counts WHERE email = ? ', (email,))
    row = cur.fetchone()
    if row is None:
        cur.execute('''INSERT INTO Counts (email, count)
                VALUES (?, 1)''', (email,))
    else:
        cur.execute('UPDATE Counts SET count = count + 1 WHERE email = ?',
                    (email,))
    conn.commit()
```

once have the emails, we select the count column in the table, where email = ? - ? is a placeholder which will be replaced by the email next to it (this is a tuple, however with only one item in it)

then we will retrieve the first one, put it in row, then if that email has not been seen (i.e. it comes back as None) then it will create a new row with that email and the count starts at 1

if the row comes back as it is in the table already, we update the count by adding 1 to it

conn.commit writes memory to disk – saves data

```
sqlstr = 'SELECT email, count FROM Counts ORDER BY count DESC LIMIT 10'

for row in cur.execute(sqlstr):
    print(str(row[0]), row[1])

cur.close()
```

once database is filled, select the table, order by count (top 10)

then execute, print each row (in a tuple, where 0 is email, 1 is count)

then close the connection

```
Enter file name:
cwen@iupui.edu 5
zqian@umich.edu 4
david.horwitz@uct.ac.za 4
louis@media.berkeley.edu 3
gsilver@umich.edu 3
stephen.marquard@uct.ac.za 2
rjlowe@iupui.edu 2
wagnermr@iupui.edu 1
antranig@caret.cam.ac.uk 1
gopal.ramasammycook@gmail.com 1
```

running mbox.txt gives us this, but also it has created the sqlite file that we can open in sqlite browser:

	email	count
	Filter	Filter
1	stephen.marqua...	2
2	louis@media.ber...	3
3	zqian@umich.edu	4
4	rjlowe@iupui.edu	2
5	cwen@iupui.edu	5
6	gsilver@umich.edu	3
7	wagnermr@iupu...	1
8	antranig@caret....	1
9	gopal.ramasam...	1
10	david.horwitz@...	4
11	ray@media.berk...	1

If you run it again, this time the table has already been created, it would add to the existing data

to see this updated table in sqlite you must hit the refresh button

assignment: use the SQLite browser to make a database, insert some data and then run a query

1) create a SQLITE database or use an existing database and create a table in the database called "Ages"

```
CREATE TABLE Ages (
  name VARCHAR(128),
  age INTEGER
)
```

2) Then make sure the table is empty by deleting any rows that you previously inserted, and insert these rows and only these rows with the following commands:

```
DELETE FROM Ages;
INSERT INTO Ages (name, age) VALUES ('Valentino', 17);
INSERT INTO Ages (name, age) VALUES ('Adnan', 23);
INSERT INTO Ages (name, age) VALUES ('Alfred', 13);
INSERT INTO Ages (name, age) VALUES ('Falyn', 29);
INSERT INTO Ages (name, age) VALUES ('Richard', 36);
```

3) once the inserts are done, run the following SQL command:

```
SELECT hex(name || age) AS X FROM Ages ORDER BY X
```

4) find the first row in the resulting record set and enter the long string

assignment 2: count messages from organizations - read the mailbox data (mbox.txt) and count the number of email messages per organization (i.e. domain name of the email address) using a database with the following schema to maintain the counts

```
import sqlite3

conn = sqlite3.connect('finalemail.sqlite')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Counts')

cur.execute('''
CREATE TABLE Counts (org TEXT, count INTEGER)''')

fname = input('Enter file name: ')
if (len(fname) < 1): fname = 'mbox-short.txt'
fh = open(fname)
for line in fh:
    if not line.startswith('From: '): continue
    pieces = line.split()
    email = pieces[1]
    domain = email.split('@')
    org = domain[1]
    cur.execute('SELECT count FROM Counts WHERE org = ? ', (org,))
    row = cur.fetchone()
    if row is None:
        cur.execute('''INSERT INTO Counts (org, count)
                VALUES (?, 1)''', (org,))
    else:
        cur.execute('UPDATE Counts SET count = count + 1 WHERE org = ?',
                    (org,))
    conn.commit()

# https://www.sqlite.org/lang_select.html
sqlstr = 'SELECT org, count FROM Counts ORDER BY count DESC LIMIT 10'

for row in cur.execute(sqlstr):
    print(str(row[0]), row[1])

cur.close()
```

makes a database called finalemail.sqlite

cursor is like a file handle, you give it instructions and then read it back

drop table basically deletes the table if it already exist so you don't overlap data, it resets so you can keep running the same script

creates a table in sql with columns org (domain) and count

opens the text file, reads by line, only choosing lines that starts with 'From', and splits each word in the line in to a dictionary

email will be the second item (i.e. 1 in dict)

split the email by @ sign so we only get the second half

the if/else loop basically says, if the domain is already counted, add 1, otherwise, add it to the table and start from 1

4.3: Databases

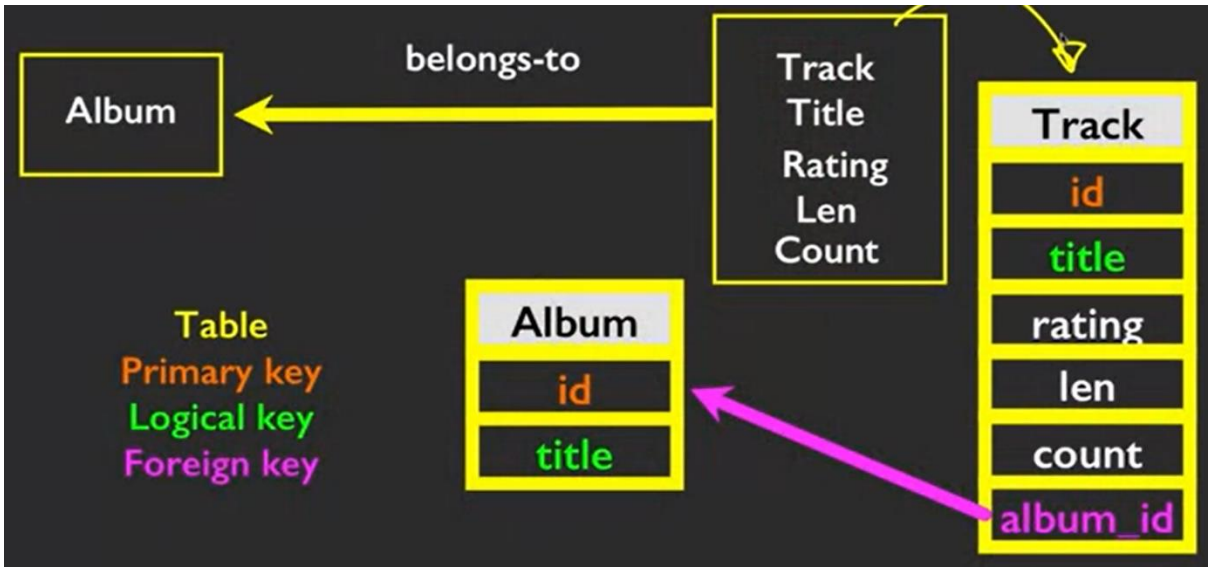
War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	★★★★★	25
Paranoid	2:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Planet Caravan	4:35	Black Sabbath	Paranoid	Metal	★★★★★	25
Iron Man	5:59	Black Sabbath	Paranoid	Metal	★★★★★	26
Electric Funeral	4:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Hand of Doom	7:1(	Black Sabbath	Paranoid	Metal	★★★★★	23
Rat Salad	2:3(	Black Sabbath	Paranoid	Metal	★★★★★	31
Jack the Stripper/Fairies Wear ...	6:1(	Black Sabbath	Paranoid	Metal	★★★★★	24
Bomb Squad (TECH)	3:28	Brent	Brent's Album			1

never put same string in more than once – use a count instead
amount of memory saved especially if this a big data set is very significant

we need to build a good data model – this user interface above is perfect, but the code we write must be efficient

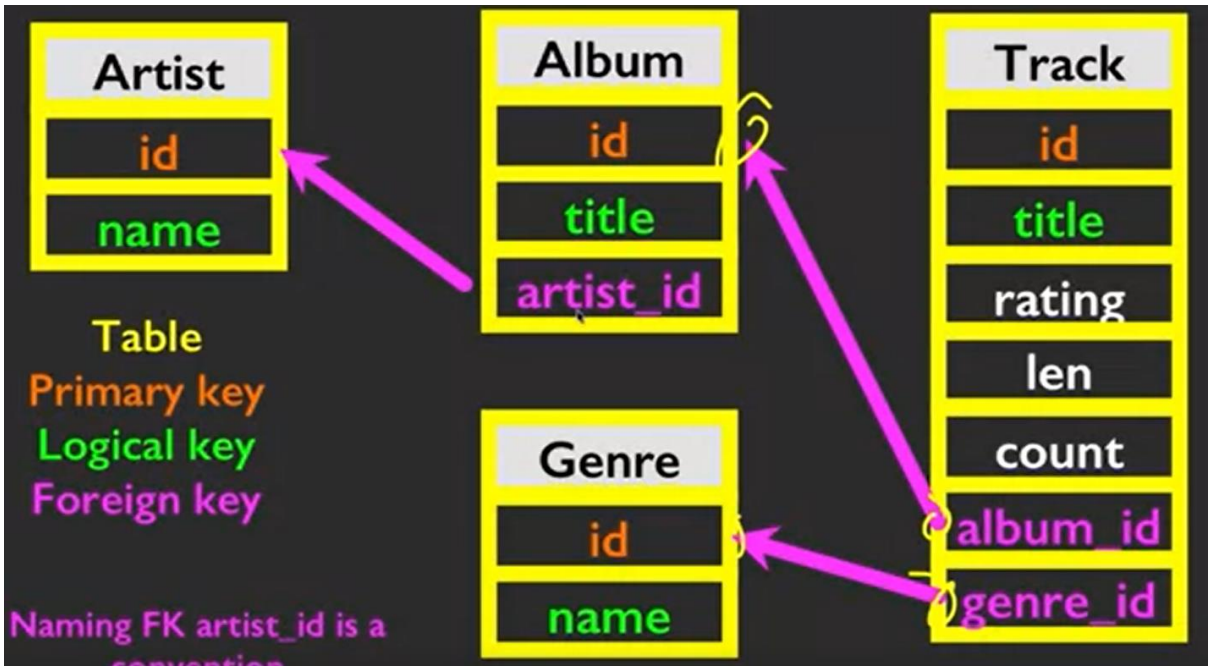
to build a good data model, start at the most important description of what the data is trying to show

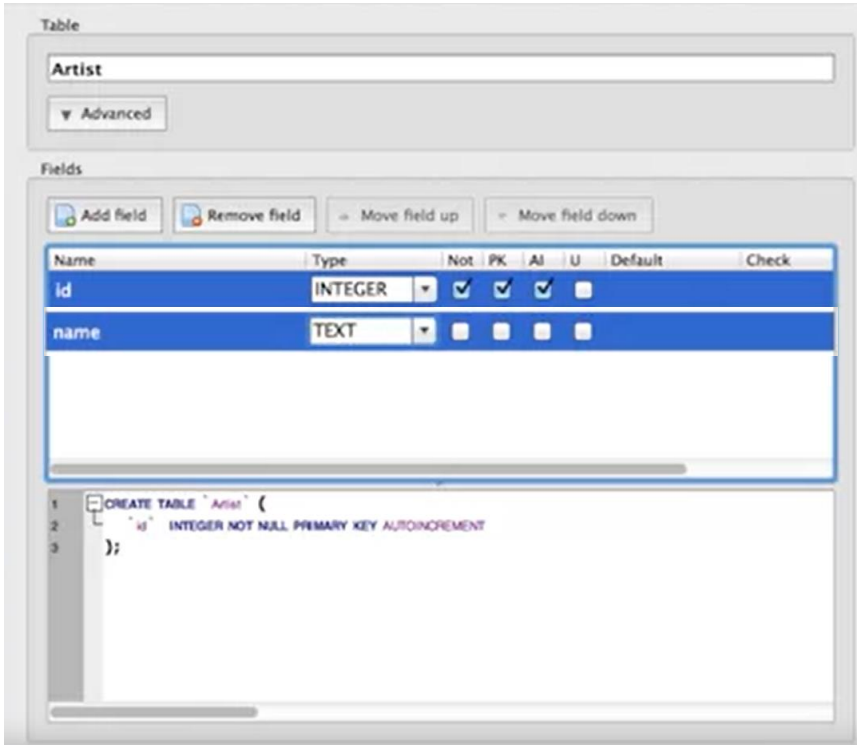
whole data about tracks – so start here
time, ratings, count are all attributes of the track
then the tracks belong to albums
albums belong to artists
last one, genre belongs to track as if you change the genre of one track the chosen user interface is set up so it does not want to change remaining tracks of the same album to the same genre



using the logical model we can now start to create tables, using keys
start with the track table, always give a primary id to identify each track, and also foreign keys can be used to point to other tables but they have to have a primary key, thus having something to actually point at

the logical key is basically things we think will be used to sort via, may be used in where clauses – important to the user





so making a table:

artist table first

id first column

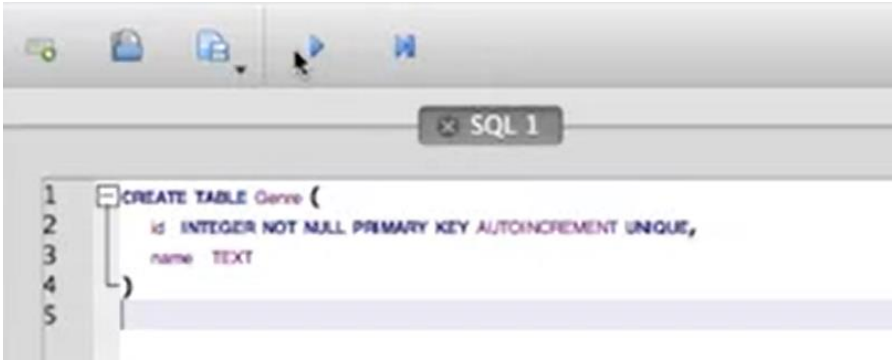
Not means not null

PK means primary key

AI means auto-increment (so computer will do it for you)

U means unsigned – means positive/negative

the first tables you make should be the ones at the end of the arrows

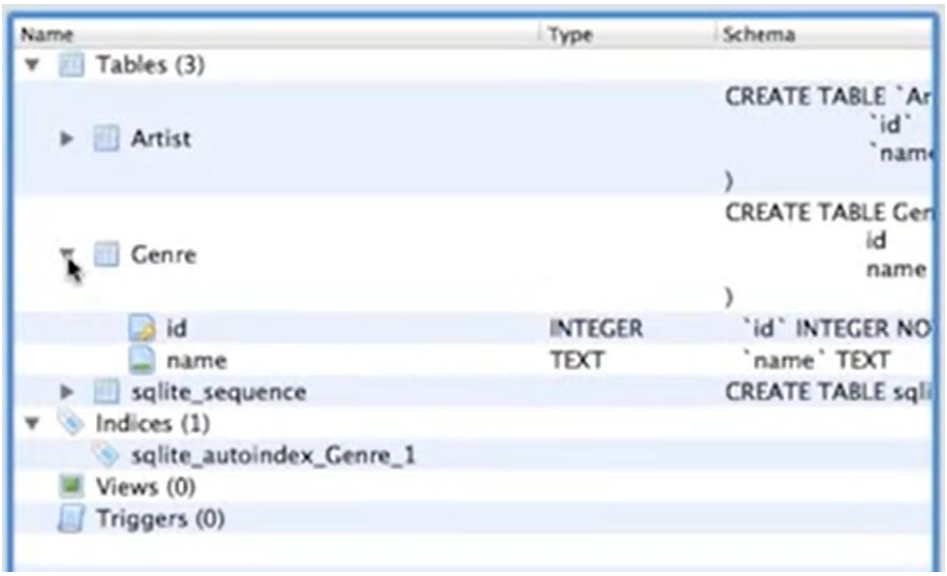


can also create tables through direct sql code, see below:

these include foreign keys

```
CREATE TABLE Genre (
        id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        name    TEXT
)
```

Genre table code



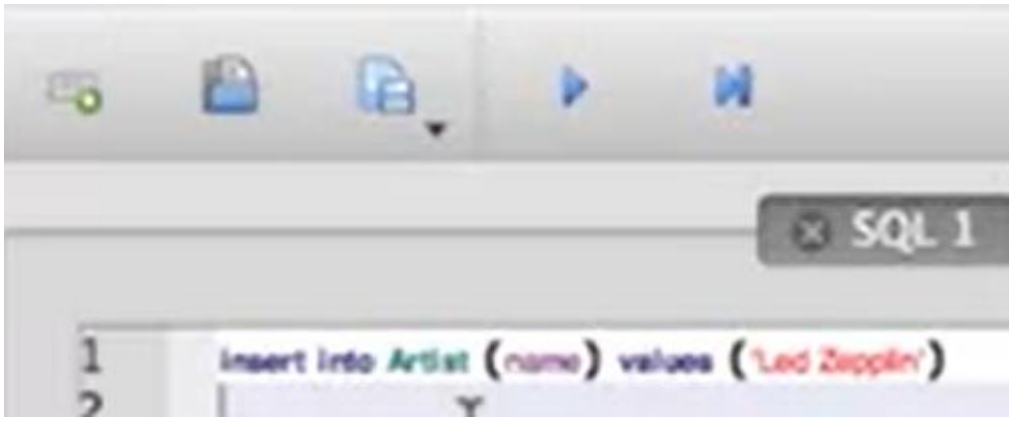
can see the result in the browse data window

```
CREATE TABLE Album (
        id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        artist_id  INTEGER,
        title   TEXT
)

CREATE TABLE Track (
        id      INTEGER NOT NULL PRIMARY KEY
                AUTOINCREMENT UNIQUE,
        title TEXT,
        album_id  INTEGER,
        genre_id  INTEGER,
        len INTEGER, rating INTEGER, count INTEGER
)
```

album table code

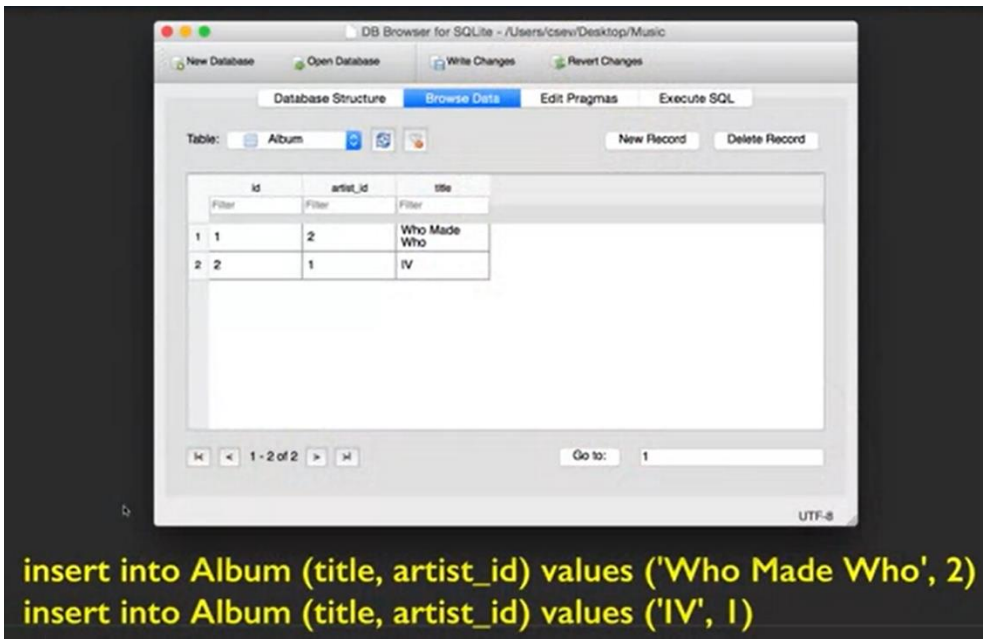
first foreign key – the artist_id

track table code



inserting data into the tables

no need to enter data for id as it is auto-generated

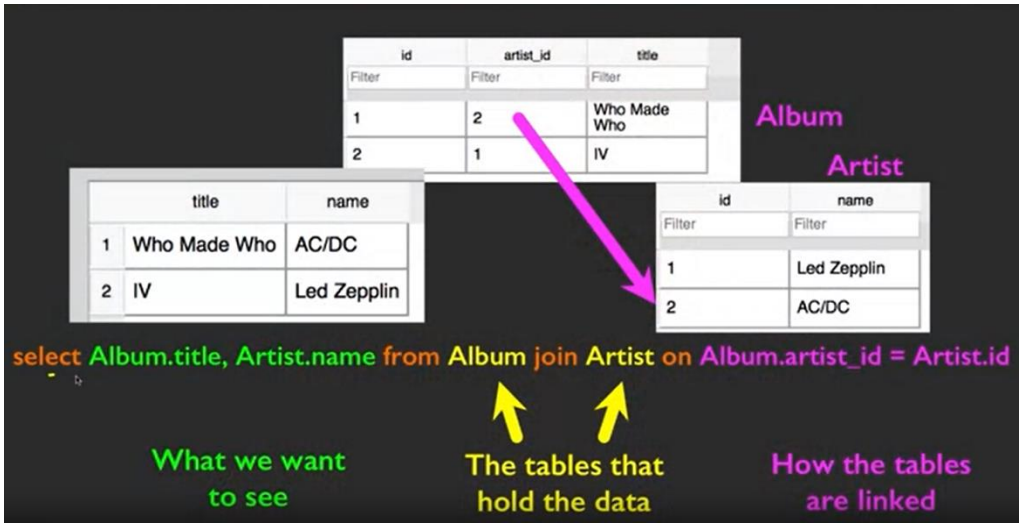


when putting albums in, we have the primary id (auto-generated) and the foreign key (we have to state which one as this key will point to a specific artist)

	id	title	album_id	genre_id	len	rating	count
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Black Dog	2	1	297	5	0
2	2	Stairway	2	1	482	5	0
3	3	About to Rock	1	2	313	5	0
4	4	Who Made Who	1	2	207	5	0

tracks have two foreign keys, pointing to album and genre

relational power is removing repeated data and replacing with references, creating a web, making reading the data much quicker



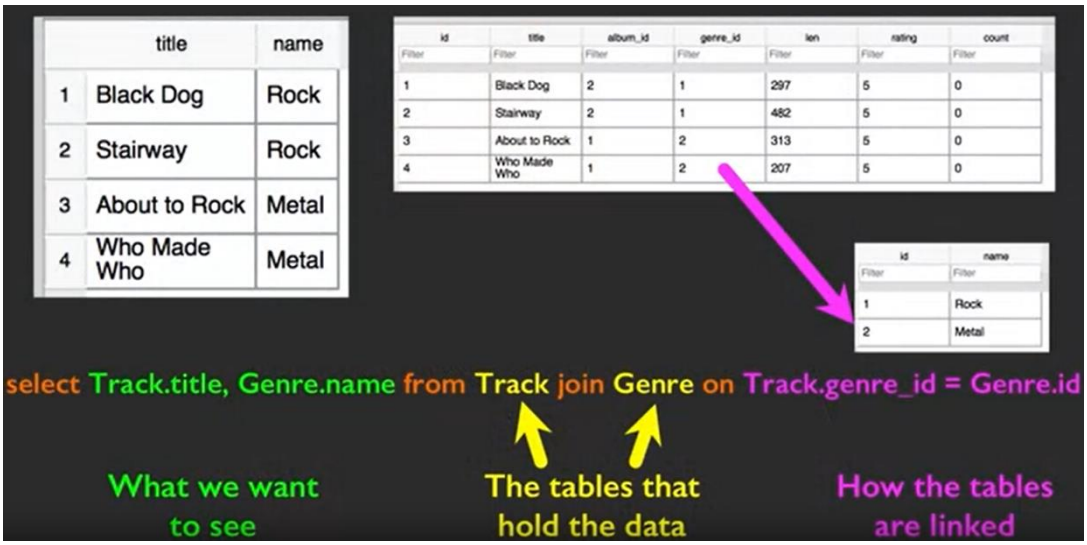
I want to make the table on left, however this is made of two separate tables

I have to write a join statement that basically says:

select basically chooses what column we want: want the row in the left table – consists of title (Album.title = this is the syntax, always table.field), then name (Artist.name = table.field, because this piece of data comes from the table Artist under name)

from Album join Artist makes a super table of both (combination)

on/where statement is linking the foreign id to primary id – essentially saying, when the foreign key artist_id inside the table Album is the same as the id in the artist table then pull that



we want table on left, we have to write the code below

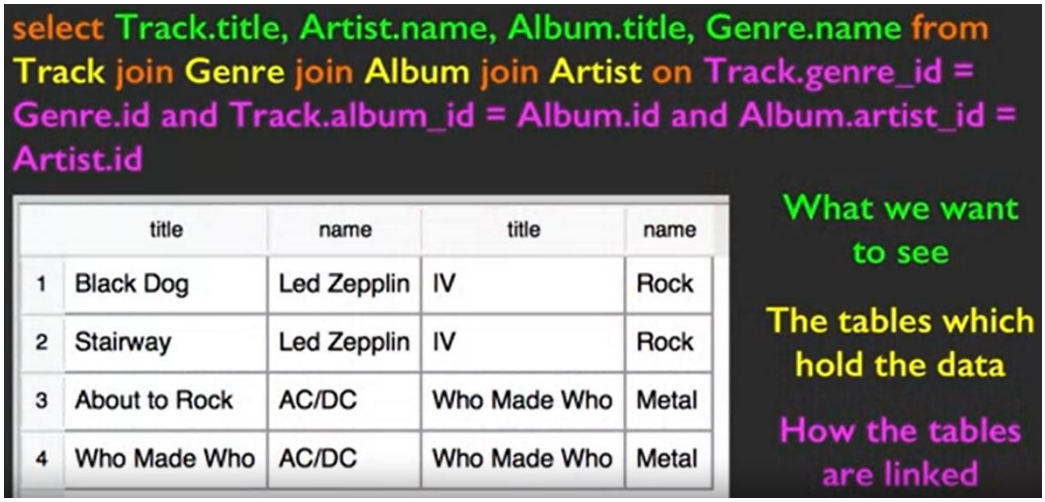
	title	genre_id	id	name
1	Black Dog	1	1	Rock
2	Black Dog	1	2	Metal
3	Stairway	1	1	Rock
4	Stairway	1	2	Metal
5	About to Rock	2	1	Rock
6	About to Rock	2	2	Metal
7	Who Made Who	2	1	Rock
8	Who Made Who	2	2	Metal

```
SELECT Track.title,
    Track.genre_id,
    Genre.id, Genre.name
FROM Track JOIN Genre
```

Joining two tables without an ON clause gives all possible combinations of rows.

always use on clauses – it links the foreign key

if we didn't use ON clauses we would get every possible combination



	title	name	title	name
1	Black Dog	Led Zepplin	IV	Rock
2	Stairway	Led Zepplin	IV	Rock
3	About to Rock	AC/DC	Who Made Who	Metal
4	Who Made Who	AC/DC	Who Made Who	Metal

use and to connect ON clauses

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Major Version</key><integer>1</integer>
        <key>Minor Version</key><integer>1</integer>
        <key>Date</key><date>2015-11-24T11:12:10Z</date>
        <key>Application Version</key><string>12.3.1.23</string>
        <key>Features</key><integer>5</integer>
        <key>Show Content Ratings</key><true/>
        <key>Music Folder</key><string>file:///Users/csev/Music/iTunes/iTunes%20Music/</string>
        <key>Library Persistent ID</key><string>B7006C9E9799282E</string>
        <key>Tracks</key>
        <dict>
                <key>369</key>
                <dict>
                        <key>Track ID</key><integer>369</integer>
                        <key>Name</key><string>Another One Bites The Dust</string>
                        <key>Artist</key><string>Queen</string>
                        <key>Composer</key><string>John Deacon</string>
                        <key>Album</key><string>Greatest Hits</string>
                        <key>Genre</key><string>Rock</string>
                        <key>Kind</key><string>MPEG audio file</string>
                        <key>Size</key><integer>4344295</integer>
                        <key>Total Time</key><integer>217103</integer>
                        <key>Disc Number</key><integer>1</integer>
                        <key>Disc Count</key><integer>1</integer>
                        <key>Track Number</key><integer>3</integer>
                        <key>Track Count</key><integer>17</integer>
                        <key>Year</key><integer>1980</integer>
                        <key>Date Modified</key><date>2006-02-14T16:13:02Z</date>
                        <key>Date Added</key><date>2006-02-14T16:12:53Z</date>
                        <key>Bit Rate</key><integer>160</integer>
                        <key>Sample Rate</key><integer>44100</integer>
                        <key>Play Count</key><integer>55</integer>
                        <key>Play Date</key><integer>3518868190</integer>
                        <key>Play Date UTC</key><date>2015-07-04T19:23:10Z</date>
                        <key>Skip Count</key><integer>1</integer>
                        <key>Skip Date</key><date>2015-10-14T23:31:47Z</date>
                        <key>Rating</key><integer>100</integer>
                        <key>Album Rating</key><integer>100</integer>
                        <key>Album Rating Computed</key><true/>
                        <key>Normalization</key><integer>1511</integer>
                        <key>Compilation</key><true/>
```

this is part of the itunes xml document we are extracting data from

can see there are key value pairs

dictionaries inside dictionaries

some values are inside the dictionary rather than paired (we will use lookup function – see later)

```
import xml.etree.ElementTree as ET
import sqlite3

conn = sqlite3.connect('trackdb.sqlite')
cur = conn.cursor()

# Make some fresh tables using executescript()
cur.executescript('''
DROP TABLE IF EXISTS Artist;
DROP TABLE IF EXISTS Album;
DROP TABLE IF EXISTS Track;
CREATE TABLE Artist (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE
);

CREATE TABLE Album (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    artist_id  INTEGER,
    title   TEXT UNIQUE
);

CREATE TABLE Track (
    id  INTEGER NOT NULL PRIMARY KEY
        AUTOINCREMENT UNIQUE,
    title TEXT  UNIQUE,
    album_id  INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
);
''')
fname = input('Enter file name: ')
if ( len(fname) < 1 ) : fname = 'Library.xml'

# <key>Track ID</key><integer>369</integer>
# <key>Name</key><string>Another One Bites The Dust</string>
# <key>Artist</key><string>Queen</string>
def lookup(d, key):
    found = False
    for child in d:
        if found : return child.text
        if child.tag == 'key' and child.text == key :
            found = True
    return None

stuff = ET.parse(fname)
all = stuff.findall('dict/dict/dict')
print('Dict count:', len(all))
```

creates 3 tables

first artist
then album (with artist_id)
then track (with album_id)

lookup function finds the correct tag

parse the data

then use findall to look into the 3rd set of dictionaries (dict/dict/dict)

```
for entry in all:
    if ( lookup(entry, 'Track ID') is None ) : continue

    name = lookup(entry, 'Name')
    artist = lookup(entry, 'Artist')
    album = lookup(entry, 'Album')
    count = lookup(entry, 'Play Count')
    rating = lookup(entry, 'Rating')
    length = lookup(entry, 'Total Time')

    if name is None or artist is None or album is None :
        continue

    print(name, artist, album, count, rating, length)

    cur.execute('''INSERT OR IGNORE INTO Artist (name)
        VALUES ( ? )''', ( artist, ) )
    cur.execute('SELECT id FROM Artist WHERE name = ? ', (artist, ))
    artist_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Album (title, artist_id)
        VALUES ( ?, ? )''', ( album, artist_id ) )
    cur.execute('SELECT id FROM Album WHERE title = ? ', (album, ))
```

entry will loop through every 3rd dictionary down

extract the things from said dictionary using lookup

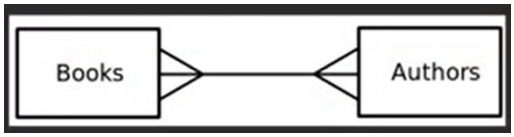
if it does not detect a name, or track id it will skip that dictionary

```
    cur.execute('''INSERT OR REPLACE INTO Track
        (title, album_id, len, rating, count)
        VALUES ( ?, ?, ?, ?, ? )''',
        ( name, album_id, length, rating, count ) )

    conn.commit()
```

the insert of ignore basically means because when the table was made it was a unique text input, if there is a repeat, ignore the entry

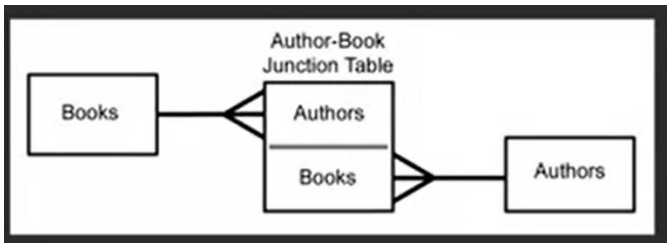
last one doesn't ignore but updates the table

4.4: Many-to-Many Relationships



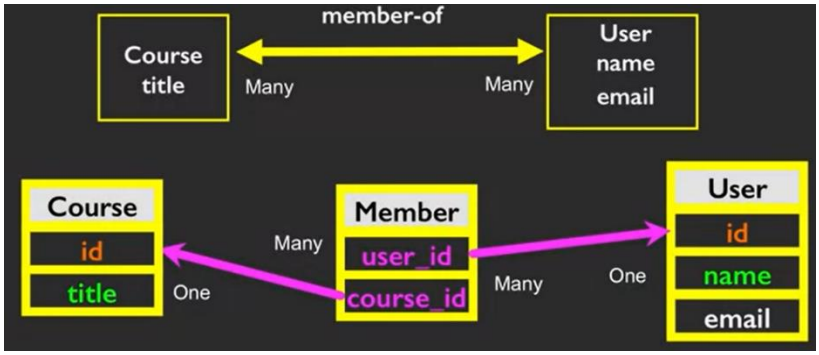
we have been looking at one-to-many relationships, a one-way sort of system

this is an example of a many-to-many relationship



you can see from the connections there are multiple lines to each box (sparrows foot) – a single connection would connect the box with only one line

can't put a foreign key because what foreign key will you try to match if there's more than one link



essentially you must break the many-to-many into multiple one-to-many relationships

the member table will have no primary keys, only two sets of foreign keys

each connection has one row that connects a particular course with a particular user

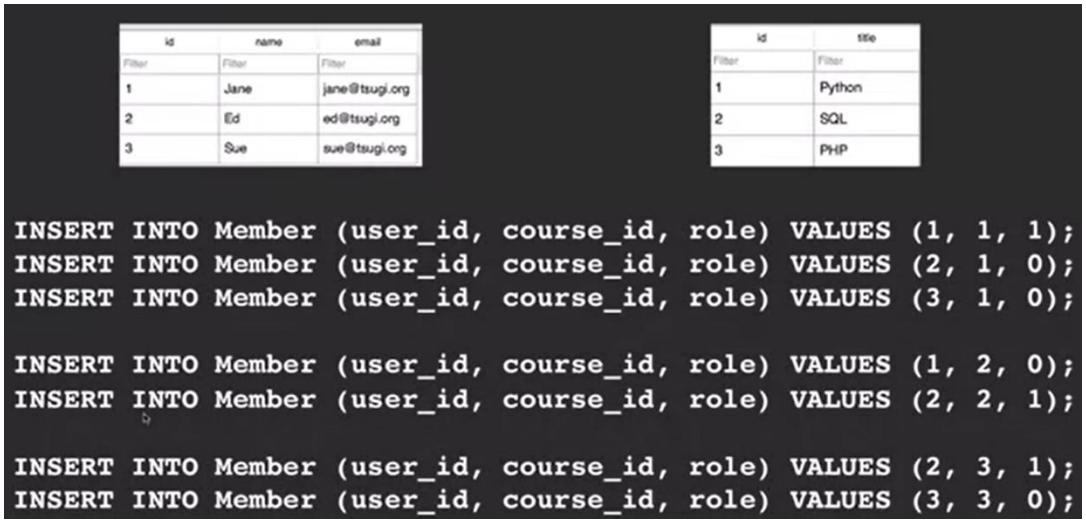
```
CREATE TABLE User (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name   TEXT,
    email  TEXT
)

CREATE TABLE Course (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title  TEXT
)

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
        role        INTEGER,
    PRIMARY KEY (user_id, course_id)
```

last table has two foreign keys, a role (which will essentially be in the middle of a user/course connection, just to state whether someone is a teacher or student) – 'on the connection'

the primary key you can see is a combination of both foreign keys, thus making the primary key unique

```
INSERT INTO User (name, email) VALUES ('Jane',
'jane@tsugi.org');
INSERT INTO User (name, email) VALUES ('Ed', 'ed@tsugi.org');
INSERT INTO User (name, email) VALUES ('Sue', 'sue@tsugi.org');

INSERT INTO Course (title) VALUES ('Python');
INSERT INTO Course (title) VALUES ('SQL');
INSERT INTO Course (title) VALUES ('PHP');
```

will insert this data in to the table



id	name	email
1	Jane	jane@tsugi.org
2	Ed	ed@tsugi.org
3	Sue	sue@tsugi.org

id	title
1	Python
2	SQL
3	PHP

```
INSERT INTO Member (user_id, course_id, role) VALUES (1, 1, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 1, 0);

INSERT INTO Member (user_id, course_id, role) VALUES (1, 2, 0);
INSERT INTO Member (user_id, course_id, role) VALUES (2, 2, 1);

INSERT INTO Member (user_id, course_id, role) VALUES (2, 3, 1);
INSERT INTO Member (user_id, course_id, role) VALUES (3, 3, 0);
```

course / user table are filled but member table is still empty

0 in role is student

	name	role	title
2	Sue	0	PHP
3	Jane	1	Python
4	Ed	0	Python
5	Sue	0	Python
6	Ed	1	SQL

```
SELECT User.name, Member.role, Course.title
FROM User JOIN Member JOIN Course
ON Member.user_id = User.id AND Member.course_id = Course.id
ORDER BY Course.title, Member.role DESC, User.name
```

Selecting the name from user table, the role from member table, the title from course table:

join them by linking the user foreign key in the member table with corresponding user id and same for the course (composite primary key)

the table at the end is ordered as such: by alphabetical order descending by course, then by member (meaning if you have the same course, the next thing to sort by xis the role, so 1 goes first as it's descending, thus showing teachers first), then the name comes after

```
cur.execute('''INSERT OR IGNORE INTO User (name)
    VALUES ( ? )''', ( name, ))
cur.execute('SELECT id FROM User WHERE name = ? ', (name, ))
user_id = cur.fetchone()[0]
```

the last line fetches one record from the cursor, essentially pulling the id linked to the name that we just entered into the record – if it is a repeat string because of the ignore part the user_id will just stay the same as the first time we went through this loop

```
conn.commit()
```

because this line means it writes everything to disk, we don't always run it as it takes longer

In the following Python code sequence (assuming cur is a SQLite cursor object),

```
cur.execute('SELECT count FROM Counts WHERE org = ? ', (org, ))
row = cur.fetchone()
```

what is the value in row if no rows match the WHERE clause?

- (●) An empty list

What does the executescript() method in the Python SQLite cursor object do that the normal execute() method does not do?

- (●) It allows multiple SQL statements separated by semicolons
- ( ) It allows embeded Python to be executed
- ( ) It allows embedded JavaScript to be executed
- ( ) It allows database tables to be created

For the following Python code to work, what must be added to the title column in the CREATE TABLE statement for the Course table:

```
cur.execute('''INSERT OR IGNORE INTO Course (title)
    VALUES ( ? )''', ( title, ) )
cur.execute('SELECT id FROM Course WHERE title = ? ',
    (title, ))
course_id = cur.fetchone()[0]
```

- ( ) A PRIMARY KEY indication
- ( ) An AUTOINCREMENT indication
- (●) A UNIQUE constraint
- ( ) A NOT NULL constraint

```
import json
import sqlite3

conn = sqlite3.connect('rosterdb.sqlite')
cur = conn.cursor()

# Do some setup
cur.executescript('''
DROP TABLE IF EXISTS User;
DROP TABLE IF EXISTS Member;
DROP TABLE IF EXISTS Course;

CREATE TABLE User (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name   TEXT UNIQUE
);

CREATE TABLE Course (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title  TEXT UNIQUE
);

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role        INTEGER,
    PRIMARY KEY (user_id, course_id)
)
''')
```

this application will read roster data in JSON format, parse the file, and then produce an SQLite database that contains a User, Course, and Member table and populate the tables from the data file

```
fname = input('Enter file name: ')
if len(fname) < 1:
    fname = 'roster_data_sample.json'

# [
#   [ "Charley", "si110", 1 ],
#   [ "Mea", "si110", 0 ],

str_data = open(fname).read()
json_data = json.loads(str_data)

for entry in json_data:

    name = entry[0]
    title = entry[1]
    role = entry[2]

    print((name, title, role))

    cur.execute('''INSERT OR IGNORE INTO User (name)
        VALUES ( ? )''', ( name, ) )
    cur.execute('SELECT id FROM User WHERE name = ? ', (name, ))
    user_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Course (title)
        VALUES ( ? )''', ( title, ) )
    cur.execute('SELECT id FROM Course WHERE title = ? ', (title, ))
    course_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Member
        (user_id, course_id, role) VALUES ( ?, ?, ? )''',
        ( user_id, course_id, role ) )

    conn.commit()
```

```
SELECT User.name,Course.title, Member.role FROM
    User JOIN Member JOIN Course
    ON User.id = Member.user_id AND Member.course_id = Course.id
    ORDER BY User.name DESC, Course.title DESC, Member.role DESC LIMIT 2;
```

The output should look as follows:

```
Zeph|si110|0
Zeid|si430|0
```

Once that query gives the correct data, run this query:

```
SELECT 'XYZZY' || hex(User.name || Course.title || Member.role ) AS X FROM
    User JOIN Member JOIN Course
    ON User.id = Member.user_id AND Member.course_id = Course.id
    ORDER BY X LIMIT 1;
```

You should get one row with a string that looks like **XYZZY53656C696E613333**.

4.5: SQL Questions

Write a query that returns all of the cities that have more than the average number of customers per city. For each such city, return the country name, the city name and the number of customers. Order the result by country name ascending.

Table definitions and a data sample are given below.

▸ Schema

▾ Sample Data Tables

Table: country

id	country_name
1	Austria
2	Germany
3	United Kingdom

Table: city

id	city_name	country_id
1	Wien	1
2	Berlin	2
3	Hamburg	2
4	London	3

Table: customer

id	customer_name	city_id
1	Cust1	1
2	Cust2	4
3	Cust3	3
4	Cust4	1
5	Cust5	2
6	Cust6	1
7	Cust7	4
8	Cust8	2

```
SELECT city.city_name, country.country_name, COUNT(customer.id) AS
num_customer
FROM city
JOIN country ON city.country_id = country.id
JOIN customer ON city.id = customer.city_id
GROUP BY city.id
HAVING num_customer > (SELECT AVG(num_customer) FROM (SELECT city.id,
COUNT(customer.id) AS num_customer
FROM city
JOIN country ON city.country_id = country.id
JOIN customer ON city.id = customer.city_id
GROUP BY city.id) AS temp)
ORDER BY country.country_name;
```

solution

[average number of customers per city I think means out of all cities what is the average – if there are 8 customers, 4 cities in total, average is 2 customers per city]

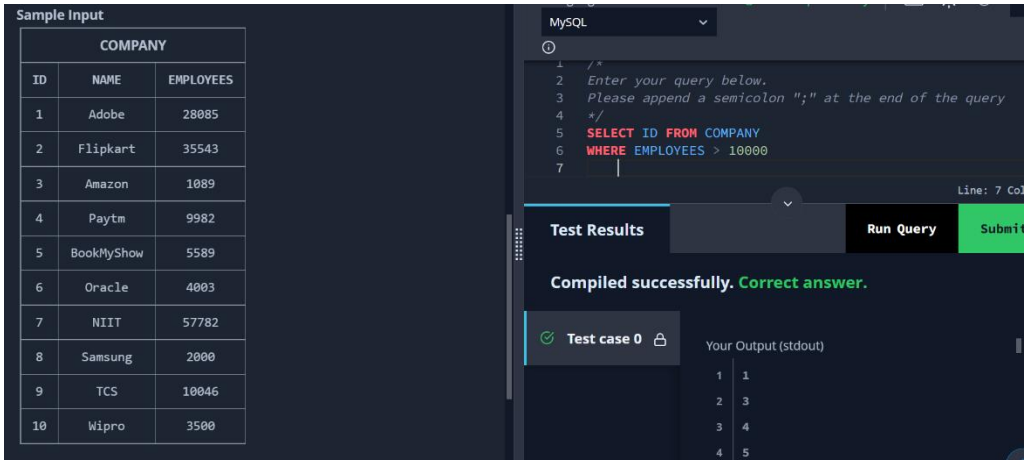


table with id, name and employees column

have to select all companies with employees over 10000 and print only the ids

customerid [PK] character	companyname character varying (40)	contactname character varying (30)	contacttitle character varying (30)	address character varying (60)	city character varying (15)	region character varying (15)	postalcode character varying (10)	country character varying (15)	ph
1 ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	[null]	12209	Germany	
2 ANATR	Ana Trujillo Emparedados y ...	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	[null]	05021	Mexico	
3 ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	[null]	05023	Mexico	
4 AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	[null]	WA1 1DP	UK	
5 BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	[null]	S-958 22	Sweden	
6 BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	[null]	68306	Germany	
7 BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	[null]	67000	France	
8 BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid	[null]	28023	Spain	

SELECT *
FROM customers

Basically opens all columns in table called customers

	country character varying (15)	count bigint
1	Argentina	3
2	Spain	5
3	Switzerland	2
4	Italy	3
5	Venezuela	4
6	Belgium	2
7	Norway	1
8	Sweden	2
9	USA	13

bunch of different countries – want to know number of customers for each country

SELECT country, count(*)
FROM customers
GROUPBY country

without groupby it will return an error – country is a non-aggregate column, combined with an aggregate function (count), therefore you need a groupby statement

can add a:
ORDER BY count(*) DESC

	country character varying (15)	city character varying (15)	count bigint
1	UK	London	6
2	Mexico	México D.F.	5
3	Brazil	Sao Paulo	4
4	Brazil	Rio de Janeiro	3
5	Argentina	Buenos Aires	3
6	Spain	Madrid	3
7	France	Paris	2
8	France	Nantes	2
9	Portugal	Lisboa	2
10	USA	Portland	2

can be even more specific:

SELECT country, city, count(*)
FROM customers
GROUPBY country, city
ORDER BY count(*) DESC

remember to add the city to the groupby clause

	country character varying (15)	city character varying (15)	count bigint
1	UK	London	6
2	Mexico	México D.F.	5
3	Brazil	Sao Paulo	4
4	Argentina	Buenos Aires	3
5	Spain	Madrid	3
6	Brazil	Rio de Janeiro	3
7	France	Paris	2
8	USA	Portland	2

SELECT country, city, count(*)
FROM customers
GROUPBY country, city
HAVING count(*) > 1
ORDER BY count(*) DESC

having clauses come after group by

```
SELECT country, city, count(*)
FROM customers
WHERE COUNT(*) > 1
GROUP BY country, city
ORDER BY count(*) DESC
```

this doesn't work as can't do filter (where) on an aggregate function

can only use having and has to be after groupby when using aggregate function (like count, or sum etc.)

	country character varying (15)	city character varying (15)	count bigint
1	Brazil	Sao Paulo	4
2	Brazil	Rio de Janeiro	3
3	Argentin	Buenos Aires	3
4	Spain	Madrid	3
5	France	Nantes	2
6	France	Paris	2
7	Portugal	Lisboa	2
8	Denmark	Århus	1
9	Canada	Vancouver	1

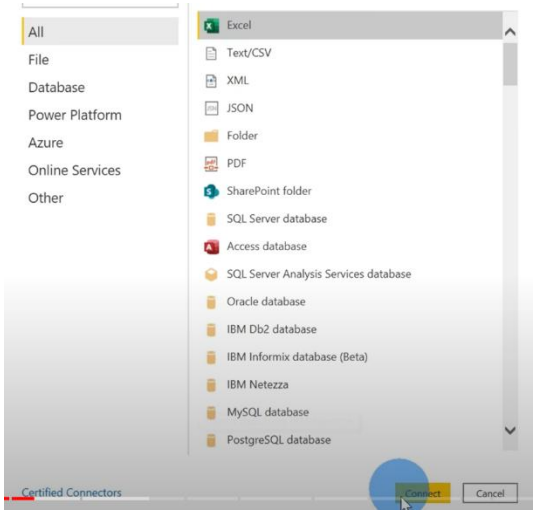
```
SELECT country, city, count(*)
FROM customers
WHERE country LIKE '%a'
GROUPBY country, city
ORDER BY count(*) DESC
```

however, this filter works with aggregate – it says only return countries with 'a' in it

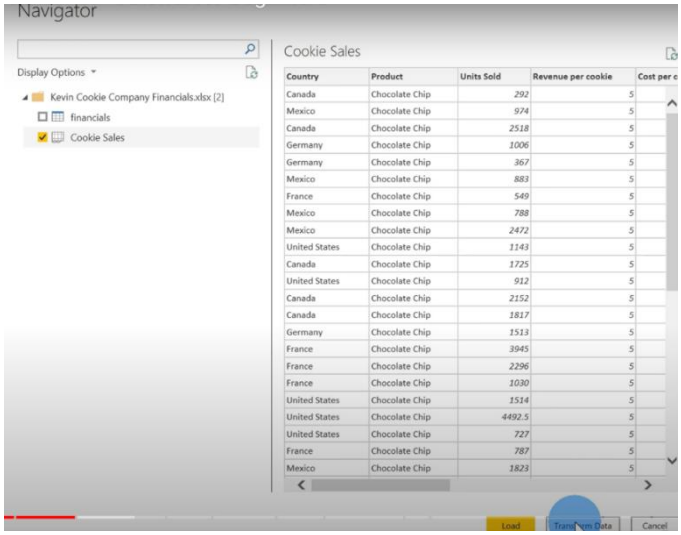
but can add HAVING clause to filter more

Chapter 5: Power Bi

5.1: Basics



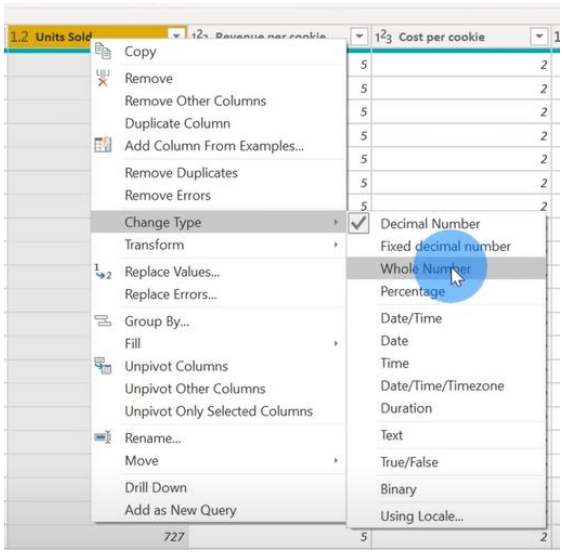
can choose whatever format of data you wanted to enter into power bi



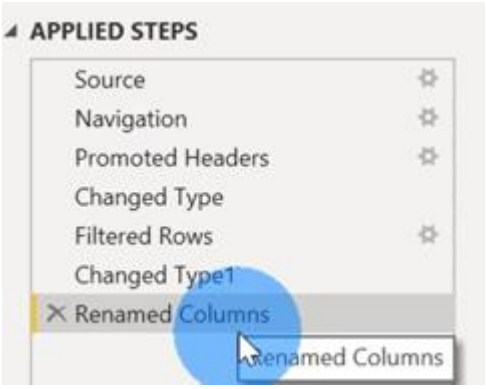
once format has been chosen you can either load the data straight or transform it (selecting what exactly from your data is brought into power bi)

opens it up into power query editor

here you can remove items from columns

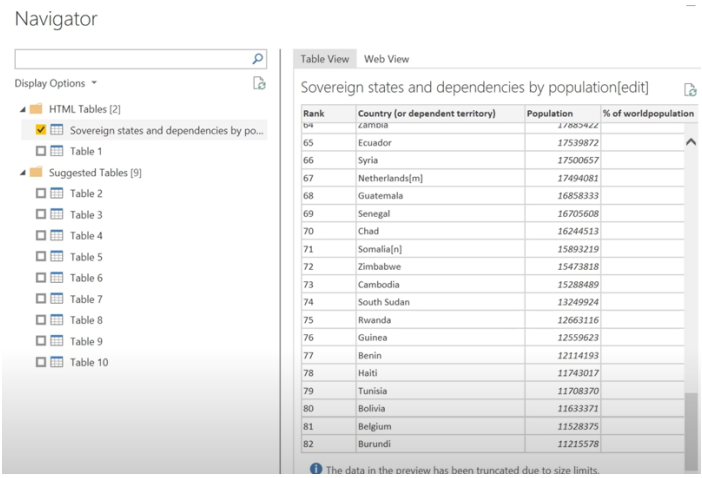


or change decimals in a column to whole numbers



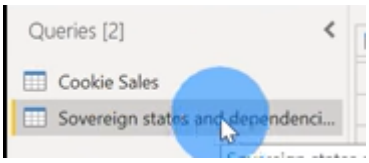
on the side panel (right side) can see all transformations done

can also undo them if deemed unnecessary

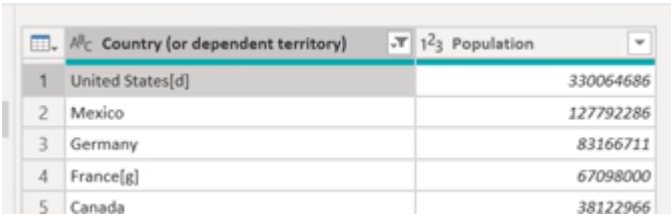


you can also pull in data from the web

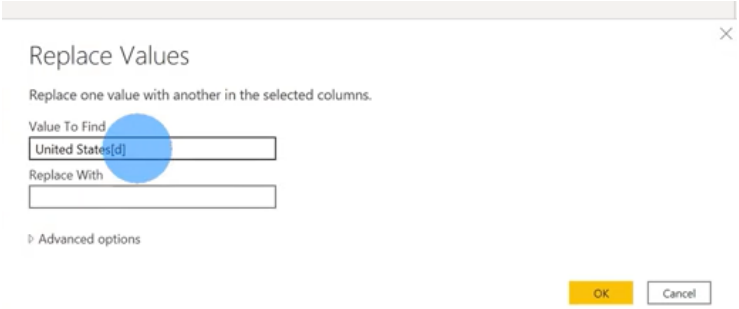
all you do is paste the url and power bi will scan the page for any data it can insert



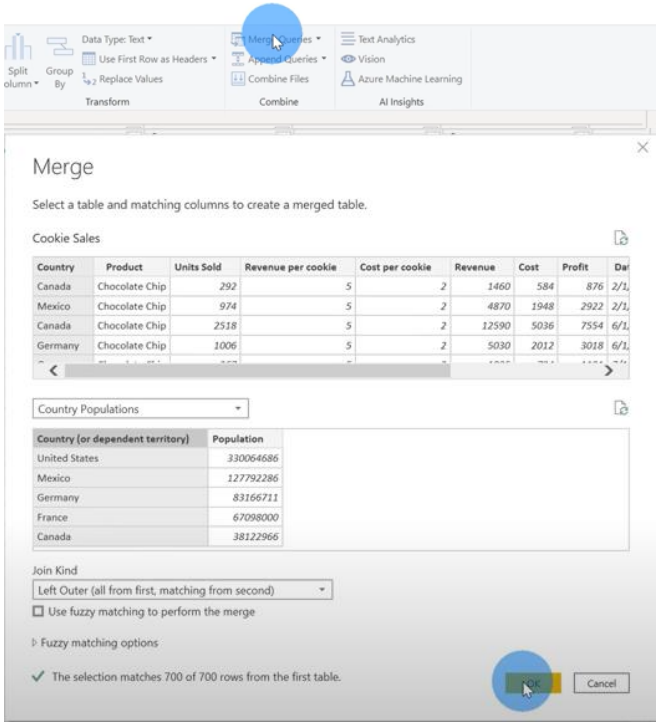
you will be able to see the different queries on the left hand side



by right clicking the column you can filter out data – even just by typing what you want and selecting it



can replace individual cells just by right clicking and selecting replace values



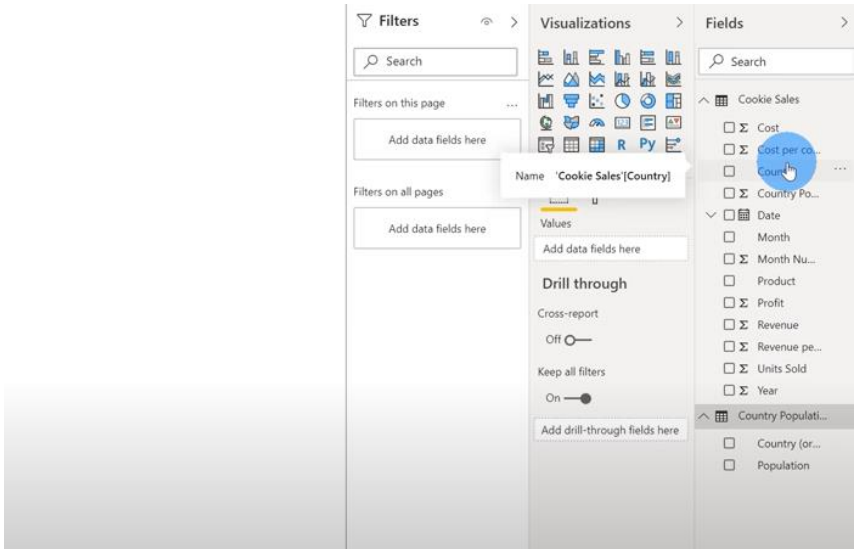
here we can merge queries

it will ask what data you want to merge – choose the columns by clicking on them

once chosen, right at the bottom it will say X/Y rows have matches

in this case we have 700/700 so we shouldn't find any blank cells

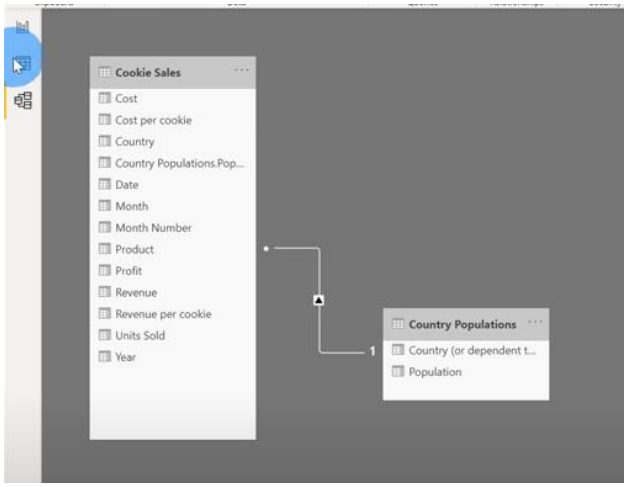
once accepted, you can change the newly merged column to show whatever data you want from the second query/table



once happy with your data you can press apply and close

this will bring you to a blank canvas with visualisation tools

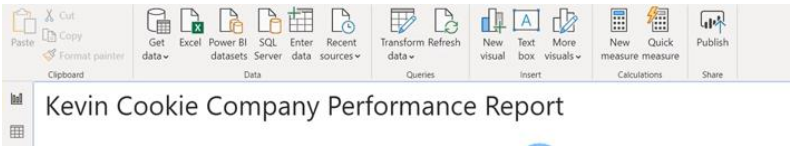
on the side you can see fields which represent all the columns from our queries



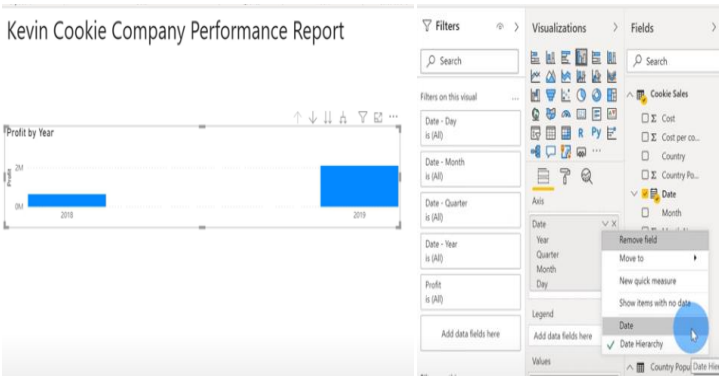
first button on left hand side is for reports – visuals go here

second button is data – similar to spreadsheet view of our transformed data
can also view individual queries if needed

third button is model – shows like a data schema



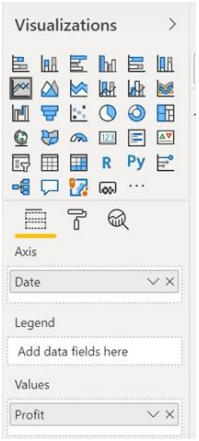
now being in report it starts blank, we can add text boxes



after choosing profit and date in our fields on the left, it automatically gives us a bar chart (which we can change)

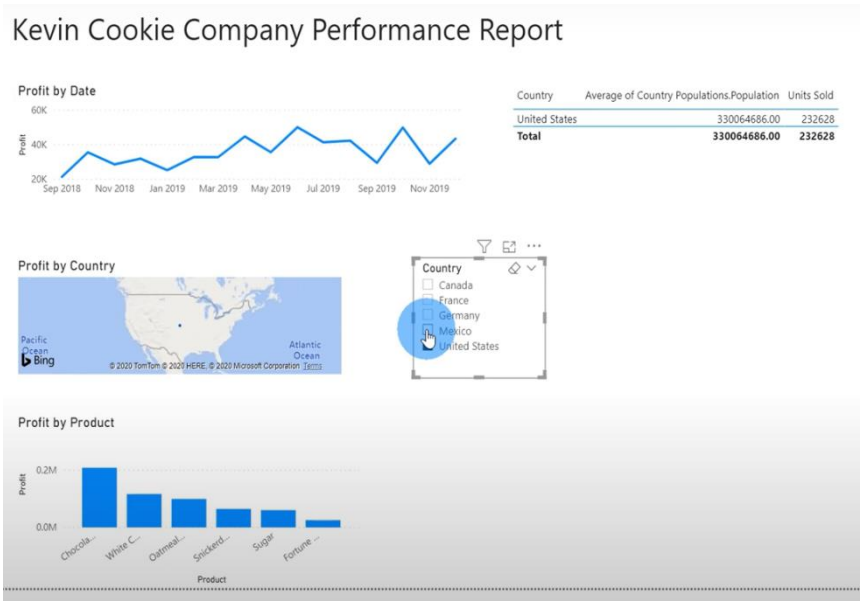
if you hover over the bars it gives more info/exact value

if you look at the menu opened on the right, we can change the settings from date to date hierarchy and then it will produce a bar chart based off months



under the visualisation tools we can choose what we see on each axis – just drag and drop to swap them around

however can also splice just by clicking a certain parameter on your existing reports/graphs



in the visualisation tools we can use a splicer as seen as the table selected

this tool allows us through one click of a button change all the reports to show only for that filter

e.g. the report showed profit/date, country/pop, profit/country, profit/product but with the splicer we can choose exactly what country we want to show data for

can format your existing tables – the paint roller icon under the visualisation tools