

Programming Idioms and Ideas: PII

- **Two kinds of loops: by-element, by-index**
 - Underneath often by index, e.g., problems when removing from a list while iterating
- **Two kinds of structured data: strings and lists**
 - Soon to add sets, tuples, dictionaries
- **Today: Strings, Lists, Sets, Oh My!**



Solving Problems, Transforming Data

- Consider the Common APT, useful in the interactive game Jotto you'll write

- "seats", "tease" -> 4
- "seats", "meaty" -> 3
- "seats", "stats" -> 4

The screenshot shows the JOTTO game interface. At the top, it displays 'YOUR SECRET JOTTO WORD' as 'MAPLE' and 'OPPONENT'S SECRET JOTTO LETTERS' as 'WNGOR'. Below this is a table with columns for 'SCORE', 'OPPONENT'S TEST WORD', 'NO. OF JOTS', 'YOUR TEST WORD', and 'NO. OF JOTS'. The table contains the following data:

SCORE	OPPONENT'S TEST WORD	NO. OF JOTS	YOUR TEST WORD	NO. OF JOTS
100	FCLASK	2	WHALE	1
95	LULLS	1	SHAKE	0
90	POUMP	3	FLING	2
85	SQUMP	3	EYUNG	2
80	LYMPH	3	SLANG	2
75	NYMPH	2	GROAN	4

- Ideas: loop over word1, cross out in word2

- 's', "*tats" 1 *does it matter which 's' ?*
- 'e', "*tats" 1 *can you replace 's' with '*'?*
- 'a', "*t*ts" 2

Ideas into code: thinking about loops

- As you loop over 's', 't' ... find and "mark"
 - You can look up the 's' in word2, find index
 - You can use index in word1 and in word2

```
for ch in word1:  
    dex = word2.find(ch)  
    if dex != -1:
```

```
for k in range(len(word1)):  
    dex = word2.find(word1[k])  
    if dex != -1:
```

Using lists rather than strings

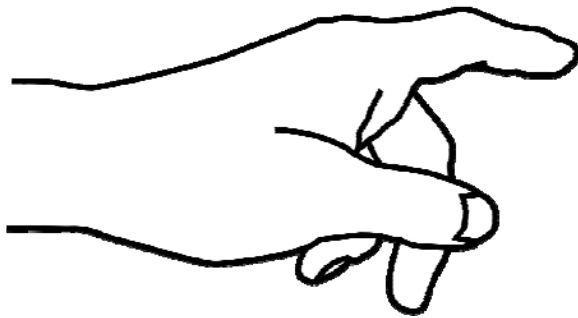
- Strings are immutable, can create new ones, but cannot change, lists are mutable!
 - Using a list instead makes code easier, *unfortunately list has no find, only index*

```
for ch in word1:
    dex = word2.find(ch)
    if dex != -1:
        word2 = word2[:dex] + '*' + word2[dex+1:]
```

```
for ch in list1:
    if ch in list2:
        dex = list2.index(ch)
        list2[dex] = '*'
```

Which loop is right? Index or Element?

- It Depends! (always a good answer)
 - If you're going to always use one loop, to avoid having to make a choice, which one to use?
 - Can you go simply from index to element?
 - Can you go simply from element to index?



STABLE																							
half life more than one trillion years																							
half life in range of billion years																							
half life in range of million years																							
half life in range of thousands of years																							
half life in range of years																							
half life in range of days																							
half life in range of hours																							
half life in range of minutes																							
half life in range of seconds																							
half life in range of milliseconds																							
no life understood																							
1 H Hydrogen																	2 He Helium						
3 Li Lithium	4 Be Beryllium																	5 B Boron	6 C Carbon	7 N Nitrogen	8 O Oxygen	9 F Fluorine	10 Ne Neon
11 Na Sodium	12 Mg Magnesium																	13 Al Aluminum	14 Si Silicon	15 P Phosphorus	16 S Sulfur	17 Cl Chlorine	18 Ar Argon
19 K Potassium	20 Ca Calcium	21 Sc Scandium	22 Ti Titanium	23 V Vanadium	24 Cr Chromium	25 Mn Manganese	26 Fe Iron	27 Co Cobalt	28 Ni Nickel	29 Cu Copper	30 Zn Zinc	31 Ga Gallium	32 Ge Germanium	33 As Arsenic	34 Se Selenium	35 Br Bromine	36 Kr Krypton						
37 Rb Rubidium	38 Sr Strontium	39 Y Yttrium	40 Zr Zirconium	41 Nb Niobium	42 Mo Molybdenum	43 Tc Technetium	44 Ru Ruthenium	45 Rh Rhodium	46 Pd Palladium	47 Ag Silver	48 Cd Cadmium	49 In Indium	50 Sn Tin	51 Sb Antimony	52 Te Tellurium	53 I Iodine	54 Xe Xenon						
55 Cs Cesium	56 Ba Barium	57 La Lanthanum	72 Hf Hafnium	73 Ta Tantalum	74 W Tungsten	75 Re Rhenium	76 Os Osmium	77 Ir Iridium	78 Pt Platinum	79 Au Gold	80 Hg Mercury	81 Tl Thallium	82 Pb Lead	83 Bi Bismuth	84 Po Polonium	85 At Astatine	86 Rn Radon						
87 Fr Francium	88 Ra Radium	89 Ac Actinium	104 Rf Rutherfordium	105 Db Dubnium	106 Sg Seaborgium	107 Bh Bohrium	108 Hs Hassium	109 Mt Meitnerium	110 Ds Darmstadtium	111 Rg Roentgenium	112 Uub Ununbium	113 Uut Ununtrium	114 Uuq Ununquadium	115 Uup Ununpentium	116 Uuh Ununhexium	117 Uus Ununseptium	118 Uuo Ununoctium						
58 Ce Cerium	59 Pr Praseodymium	60 Nd Neodymium	61 Pm Promethium	62 Sm Samarium	63 Eu Europium	64 Gd Gadolinium	65 Tb Terbium	66 Dy Dysprosium	67 Ho Holmium	68 Er Erbium	69 Tm Thulium	70 Yb Ytterbium	71 Lu Lutetium										
90 Th Thorium	91 Pa Protactinium	92 U Uranium	93 Np Neptunium	94 Pu Plutonium	95 Am Americium	96 Cm Curium	97 Bk Berkelium	98 Cf Californium	99 Es Einsteinium	100 Fm Fermium	101 Md Mendelevium	102 No Nobelium	103 Lr Lawrencium										

Eating Well or Good Eating: APT

- <http://www.cs.duke.edu/csed/pythonapt/eatinggood.html>
- **First think about solving this by hand...**
 - In translating to Python, what's easy? Harder?
 - Can we find diners who eat at Elmo's easily?
- **Structure**
 - Strings and lists
 - Using `.split(...)`



Eliminating Duplicates

- **Could process a list, avoid double counting by checking, but much easier solution: set!**
 - **Part of Python and many other languages**
 - *Typically implemented to be very efficient in determining membership*
- **Set – collection like list, but not indexable**
 - **Can `.add()`, `.remove()`,**
 - **Can iterate, cannot slice**
 - **Can `if foo in coll:` where `coll` is set or list**

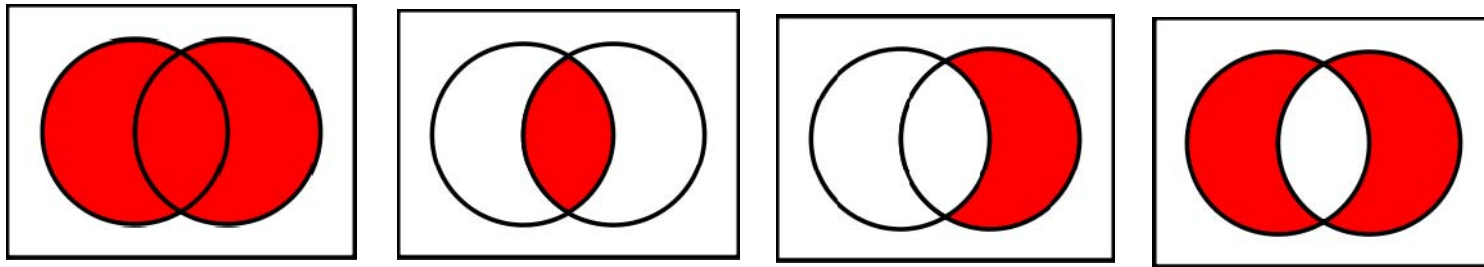
Thinking about sets

- Use `list.append(x)`, use `set.add(x)`
 - If already in set, nothing happens

- Can create set from a list all at once

```
uni = set([1,2,3,1,2,3,1,2,3,1,1,2,2,3,3])
```

- Later we'll see union `|`, intersection `&`, difference `-` and other operations ^ TBDiscussed



Question Interlude

<http://bit.ly/101fall15-1008-1>

Summary (from wikibooks)

- `set1 = set()` # A new empty set
- `set1.add("cat")` # Add a single member
- `set1.update(["dog", "mouse"])` # Add several members
- `set1.remove("cat")` # Remove a member - error not there

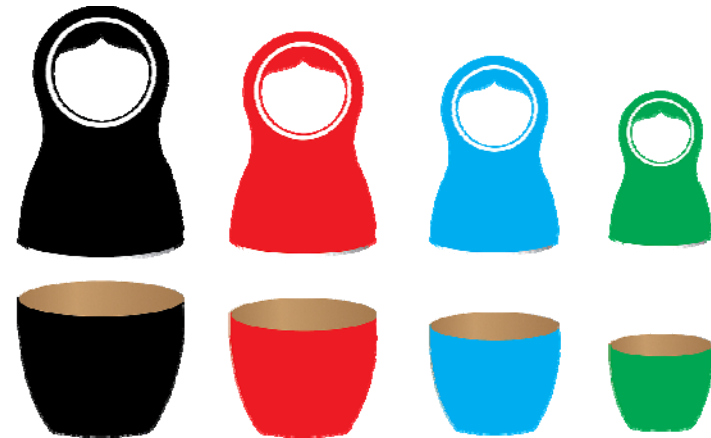
- `for item in set1:` # Iteration or "for each element"
- `len(set1)` # Length, size
- `isempty = len(set1) == 0` # Test for emptiness
- `set1 = set(["cat", "dog"])` # Initialize set from a list

- `set3 = set1 & set2` # Intersection
- `set4 = set1 | set2` # Union
- `set5 = set1 - set2` # Set difference
- `set6 = set1 ^ set2` # Symmetric difference (elements in either set but not both)

- `Is Subset: set1 <= set2` # Subset test
- `Is Superset: set1 >= set2` # Superset test
- `set7 = set1.copy()` # shallow copy (copies set, not elts)
- `set8.clear()` # Clear, empty, erase

Indexes within indexes, loop in loops

- **Very useful in solving two-dimensional and other problems**
 - **Lists are one-dimensional, for example**



List in a list and loop in a loop

- `z = [[1,2,3], [4,5,6], [7,8,9]]`
 - for `x` in `z`: what is type of `x`?
- Use one loop inside another to access both
 - Could be list of student info as well

```
for x in z:  
    for y in x:  
        #what type is y?
```



Looping with Indexes

- **How to understand a loop-in-a-loop?**
 - **What changes in the inner loop**

```
def doublenest(n):  
    for i in range(n):  
        for j in range(n):  
            print i,j
```

```
def doublenest2(n):  
    for i in range(n):  
        for j in range(i+1,n):  
            print i,j
```

Create "couples"

- **Aname is fixed as the inner loop executes**
 - See output to reinforce this idea

```
A = ['sam', 'lou', 'chris']  
B = ['terry', 'brook', 'val']  
for aname in A:  
    for bname in B:  
        print aname, ", ", bname
```

```
sam , terry  
sam , brook  
sam , val  
lou , terry  
lou , brook  
lou , val  
chris , terry  
chris , brook  
chris , val
```

