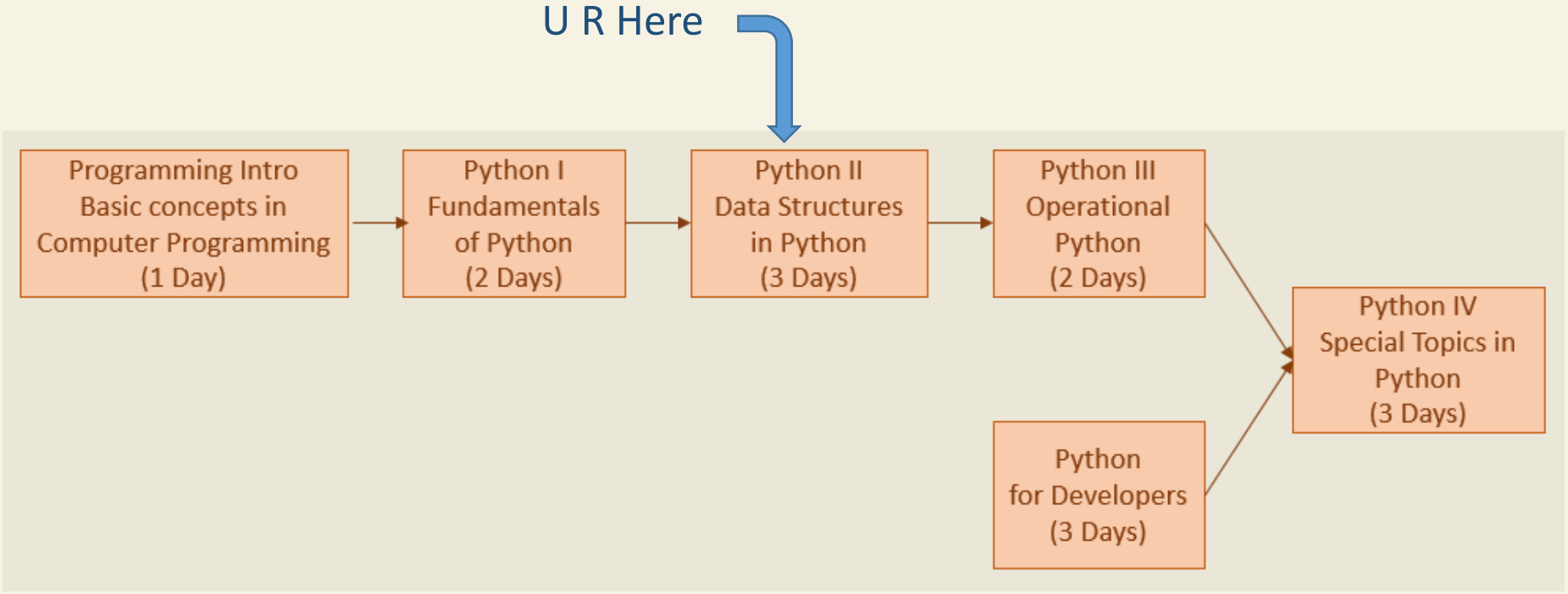
Python II
Welcome to Programming

Rev 10/6/2019

Python Classes

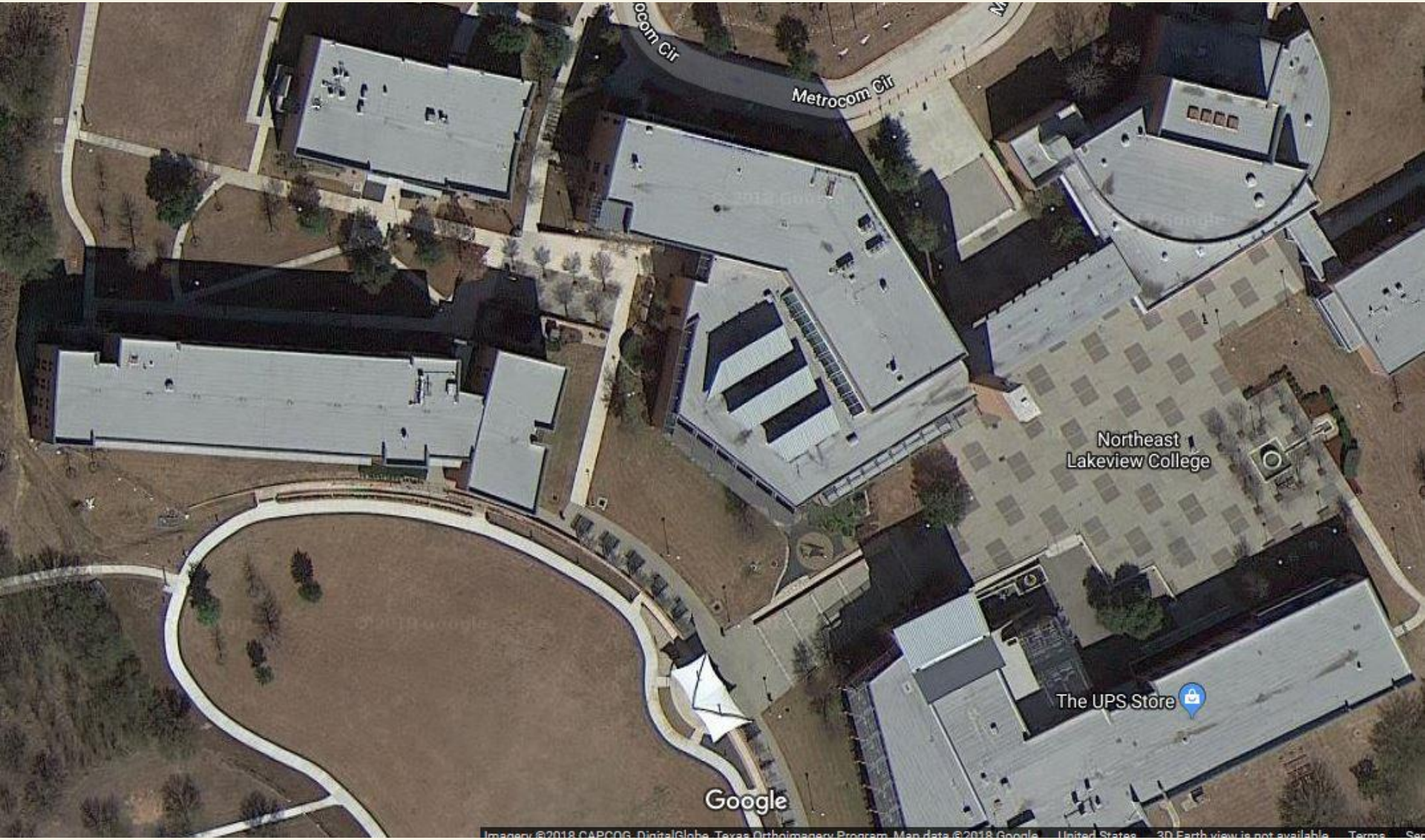


Objective:

- 20% Lecture
- 80% Lab
- Using Python V3

INTRODUCTIONS

- Instructor
- Students
  - Did you attend Python I?
  - What have you done since then?
  - How do you plan to use Python?


Metrocom Cir
Northeast
Lakeview College
The UPS Store
Google
Imagery ©2018 CAPCOG, DigitalGlobe, Texas Orthoimagery Program, Map data ©2018 Google
United States
3D Earth view is not available
Terms

PAPERWORK

- Grant registration
  - Fill in at least the fields for gender, birth date and your name at the top and the signature line at the bottom. The rest is optional, but the ACCD would appreciate having it.
- Grant daily sign-in and sign-out sheet
- If you have questions, my e-mail is:
  - weastridge@gmail.com
  - Make the subject: Python

Castle Lab Environment

- ID – student, password - student
- IDLE Install
  - For CentOS: sudo yum install python3-tools
  - For Ubuntu et al: sudo apt-get install idle3
  - Windows and MacOS come with IDLE installed
    - Macs with retina displays may require a larger font size than the default to make underscores show properly.
- CentOS7 at the Castle has IDLE already installed.
- To create an icon for IDLE:
  Drag the file Idle3.desktop to the desktop, double-click it and trust it. This is your icon for IDLE.
- Data
  - go to bit.ly/pyhandouts
  - This file contains documents, samples and data for labs

NLC Lab Environment

- Lab machines are running Windows 10 and Python 3.7
- To get to IDLE:
  - Search on Python
  - Right click on IDLE and save on Start menu.
  - Drag from Start menu to Desktop if you want and icon there.
- Do not lock these computers.
  - You cannot unlock them.
  - They have to be rebooted and you lose everything.

Class Structure

- We will use IDLE or Vim and command line. Remote users with Python installed locally will use that environment.
- Outline – Data Structures
  - Review Python I
  - Strings – operations, methods and concepts of iterable and mutable
  - Lists and tuples – operations, methods and concept of tables
  - Dictionaries – concept of quick, direct access to data
  - Sets – concepts of inclusion, exclusion and uniqueness

Python I Review

- What is the first thing you do when writing a program?
- When writing a program, what should you always have?
- What is a program failure caused by? What is the solution process called?
- What is source code? Object code? Byte code?
- What are three categories of problems you can encounter when running your program?
- Name three possible run-time exceptions.
- Python is a scripted language. Name two more.

Python I Review

- What is an identifier versus a variable? What two things does a variable point to?
- What characters can be used to form a variable name?
- Are the variables x and X the same thing?
- In Python version 2, what is the difference between 7/3 and 7//3? In Python version 3?
- If x and y are strings, is x + y valid? x * y?
- Name several built-in and imported functions you have used.
- What is an assignment statement? An expression?
- What is a Docstring? Where is it most commonly used?

Python I Review

- How is a comment created? Where can they be placed?
- Have you scanned the contents portion of PEP 8 for items you should read?
- Does a print function always cause a linefeed?
- What are the five basic data types we studied?
- What numbering system (base) does the computer use internally?
  - What is the decimal number 138 in this system? How many bits does it require?
  - How is that number represented in hexadecimal (hex)?
  - What is the hex representation of a newline character (see the ASCII chart)?
- What formatting letters did we use in Python I? There were 3.

Python I Review

- What characters can be used to enclose a string?
- What is the pydoc/pydoc3 command used for?
- What is the basic purpose of a function? Where in the program is it usually located? Must it be there?
- What is the difference between global and local variables?
  - Where did they occur in Python I?
- What statements trap exceptions? What is a bare except?
- Before an external file is read, what must be done?
- What two ways did we discuss to read the records in a file?

Python I Review

- How did we determine a complete file had been read?
- When we read numeric data from a file or the keyboard, how was it received by our program? What data type was it?
- What is whitespace? Did any whitespace characters at the end of a record interfere with int() or float() conversions?
- What format allowed us to print the whitespace characters such that we could see them?
- When opening a file, how do you determine what operations are permitted?
- If we write to an existing file, what happens to the original data?
- The print statement causes a linefeed by default. Does file.write()?

Python I Review

- Preparing for Python II (See Python for Everybody)
  - Have you discovered how to index into a string? To loop through a string?
  - In your reading, have you discovered what a list is? A tuple? A dictionary?
  - What does it mean that a list is mutable? Is a string mutable?

Review - Python Help

- help() in the interactive shell. >>>
  - help(int) built-in function
  - help('random') importable module
  - help('random.randint') function in importable module
- At bash prompt: pydoc3
  - Uses docstrings from source code
  - pydoc3 int, pydoc random, pydoc random.randint
  - pydoc3 –k string: searches for string in the synopsis lines of all available modules
- At powershell prompt: python –m pydoc ---------
- Google! When you end up in stackoverflow.com, what is the most important thing you should watch for?

Review LAB – Dice Roll

- Create a program that calls a function that simulates the rolling of a pair of dice.
- Your main program will deal with the total of the two dice.
- The rules are as follows:
  - On the first roll, a total of 7 or an 11 is an automatic win
  - On the first roll, a total of 2, 3 or 12 is an automatic loss
  - Any other number is called the Point.
  - Keep rolling the dice until one of the following occurs:
    - You roll a 7 which is a loss
    - You roll the Point number again which is a win.
- You start with $100 and bet $10 on each play.
- Print all the rolls and whether you have won or lost on one line.
- Print the funds balance and a request to play again on the next line. A 'y' or 'Y' means play again. Anything else ends play. A balance of $0 ends play automatically.

Lab Results Example

```
Beginning Balance = $100
7 You win!
Balance = $110 - Play again? y/n: y
10 7 You lose!
Balance = $100 - Play again? y/n: y
8 6 5 9 8 You win!
Balance = $110 - Play again? y/n: y
11 You win!
Balance = $120 - Play again? y/n: y
8 7 You lose!
Balance = $110 - Play again? y/n: n

Number of plays - 5
Ending Balance = $110
```

Topics – Data Structures

- **Strings** – parsing strings to locate and/or transform the data
- **Lists/Tuples** – all languages have mechanisms for tables. Lists and tuples provide this capability in Python.
- **Dictionaries** – are indexed by *keys* *and* used for quick lookup and retrieval.
- **Sets** - are an unordered collection used for membership testing/modification and eliminating duplicate entries.
- All of the above structures plus files are iterables which makes them perfect for use in 'for' loops as you will see.

Data Structure Characteristics

- Strings - A string is an ordered collection of Unicode characters. We deal mostly with ASCII; but, never forget, the international character set is very large and complex.
- Lists/Tuples - These are essentially tables/arrays. Both are ordered collections of objects. Mostly, they contain numbers and strings, but they can contain any object type at all, even other lists or tuples.
- Dictionaries - These are called maps or mappings in other languages. The idea is that you use a key for quick access to some associated data. The associated data can be as simple as a counter but it can be any object at all. As of Python 3.6, dictionaries maintain insertion order.
- Sets - These are unordered collections of specific data types; mostly numbers, strings and tuples. They follow the mathematical rules of sets and, as such, are very powerful.

Completing Labs

- We will be doing a lot of labs.
- Many people going through this class are unable to complete all of these labs.
- Actually completing the lab is not as critical as making a serious effort to do so.
  - This will make the explanation of the completed lab make more sense to you.
- An inability to finish the labs is no reason to become discouraged. This happens to most people going through this class.

Strings – Basic Operations

- Strings are sequences. As such they are ordered and support:
  - The len() function.
    - x = 'himalayas'
    - len(x) produces 9
  - The in operator - if 'yas' in x: produces a True result
- "String comparison operators (See the ASCII chart)
  - "abc" < "xyz" (True)
  - "ABC" == "abc" (False)
  - "abc" < "ABC" (?) "abc-" > "abc_" (?) "abc1" > "abc" (?)
- Using the ASCII chart, how do I test one character to see if it is:
  - a lower case letter?
  - an upper case letter?
  - a digit?

Strings - ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Iteration and Iterables

- ***Iteration*** is a general term for taking each item of something, one after another. Any time you use a loop, explicitly or implicitly, to go over a group of items, that is iteration.
- To ***iterate*** *over* a collection means to visit each element of the collection, and do some operation for each element.
- In Python, we say that a value is an ***iterable*** when your program can iterate over it. In short, an ***iterable*** represents a collection of one or more values. (e.g., in a string each value is a character)
- One of the most common ways to inspect an iterable is with a ***for*** statement, where you want to perform some operation on a collection of values.

Strings – Iteration

- x = 'himalayas'
- Strings are iterables.
- Traverse a string by iteration:

```
for j in x:
    print(j)
```

- Essentially, j becomes each character in the string.
- Try this code in the shell.
- What changes would you make to print all the letters except a's?

```
for j in x:
```

```
j          'himalayas'
Now j = 'h'
```

```
j          'himalayas'
Now j = 'i'
```

... and so on

j is a separate variable. It is NOT a pointer into the string x. If you change j, you don't change the string x.

Strings

- What does this code do?

```
word = input("enter a word: ")
letter_to_find = input( "enter a letter to find in the word: ")
count = 0
for letter in word:
    if letter == letter_to_find:
        count += 1
print("found", count, "occurrences of", letter_to_find)
```

LAB 02a

In your data folder, you will find a file containing the text for the book, "Alice in Wonderland." Read the entire file into memory. Using only the tools we have covered so far, scan this text counting all of the letters regardless of case. Keep a separate count for all occurrences of the letter 'e' again regardless of case. At the end, print the total of all letters, the number of e's and the percent of the total that the e's comprise.

Answers:

```
Total letters: 107,720
Total e's: 13,575
12.6% of all the letters are e's
```

More String Operations – Slicing

- x = 'himalayas'
- String Slicing - string[start:stop:step]
  - start – the first or only item we want (starting from zero)
  - stop – the first item we don't want
  - step – an increment other than 1 which is the default
    - x[0] is 'h', x[3] is 'a', x[6] is 'y',
    - x[1:3] is 'im', x[0:3] is 'him, where start defaults to zero if not specified.
    - x[6:] is 'yas', where stop defaults to include the last character

Strings – Slicing

- x = 'himalayas'
- Slices can be negative.
  - x[-1] is 's', x[-3] is 'y', x[-9] is 'h'
  - x[:-1] (or x[0:-1]) is 'himalaya', x[-6:-1] is 'alaya'
- Mixed notation - x[2:-2] is 'malay'
- Steps – x[2:8:2] is 'mly' You do not get the end point!
- To create a reversed string or do a reversed iteration:
  - y = x[-1::-1] or for y in x[-1::-1] : (Try these in the shell)
- x[9] is what?
- Sample – a1StringSlice.jpg

Strings

- What does this function do?

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index += 1
    return -1
```

Files as Iterables (Review)

- Remember from Python I - Files ARE iterables!
- Each record separated by a line feed (\n) can be read with a "for" statement.
- You do not need to test for end of file.
- Generic example:
  - fin = open('path to file/filename', 'rt')

    for linein in fin:
        process the record
    statements to execute upon file completion.
  - Sample - a2Read_File2.jpg

LAB 02b

Review the file tmpprecip2012.dat. It is laid out as follows:

Columns	Content
1 – 2	Month
3 – 4	Day
5 – 8	Year
9 – 13	Precipitation in the format dd.dd (inches)
14 – 16	High Temperature

The file contains San Antonio temperature and precipitation data for 2012. Accumulate the number of days with measurable precipitation and the precipitation total for the year and print them.

Answers:

```
Rain days - 72
Rain amount - 39.23000000000001 inches
```

Methods

- Methods are essentially functions.
- Calling a function
  - You simply invoke the name of the function followed by parentheses containing any (or no) arguments.
  - The function is not attached to any other object.
- Calling a method. General form: object.method()
  - The method must be invoked by attaching it to an appropriate object using dot notation (e.g., open_file.write(line) where write is the method name and line is a variable containing a string.)
  - With a method, there is one more argument passed; the object the method is attached to.
    - In the above example, two arguments are passed; open_file and line, in that order.
    - This is the only difference between a function and a method.

String Methods

- Remember, functions can be fruitful or void (Result of None)
- All of the string methods (functions) are fruitful.
- As with all methods, string methods must be attached to the object type for which they were constructed.
- Strings cannot be changed in place. (Immutable)
  - Every time you make a valid change to a string, a new string is created in a new location.
  - Some of the data structures we will study will be mutable and their method may operate very differently.
- See Python Notes for a complete list of string methods. We will use only a relatively-small subset of them.

Strings - Methods

- Invoked with dot notation
  - x = 'himalayas'
  - x.upper() returns 'HIMALAYAS'
  - x.find("ya") returns 6
  - x.find("az") returns -1
  - x.count("a") returns 3
  - x.startswith("hi") returns True
  - x.endswith("az") returns False
  - x.isalpha() returns True
  - x.isdigit() returns False
- See sample bStringMethods1.jpg

LAB 02a Revisited

In your data folder, you will find a file containing the text for the book, "Alice in Wonderland. Read the entire file into memory. Using only the tools we have covered so far, scan this text counting all of the letters. Keep a separate count for all occurrences of the letter 'e'.

Review the program you produced for this lab and use the string methods we have covered to simplify the code.

Answers:

```
Total letters: 107,720
Total e's: 13,575
12.6% of all the letters are e's
```

Strings

- Built-in functions chr() and ord().
- ord() returns the numeric equivalent of a unicode character.
- chr() returns the unicode character corresponding to the integer provided.
  - x = 'himalayas'
  - mynum = ord(x[3]) returns the integer 97 as mynum
  - letter = chr(mynum) returns the character 'a'

LAB 02c

Using the chr() and ord() built-in functions, print the ASCII characters corresponding to the numbers 32 through 126. Then, from the string module, import the variable printable. If necessary, review the string module through help('string') in the shell or pydoc[3] string from command line. Afterwards, iterate through the "printable" string and print out each entry and its corresponding ordinal. Be sure to use !r (or the older %r) formatting on the characters in printable items so you can see the whitespace characters. My solution to Lab 02b contains an example of !r.

See the next slide for expected output.

LAB 02c Results

The character representation of numbers 32 through 126

```
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < =
> ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [
\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y
z { | } ~
```

The ordinal of each character in the printable variable

'0'	48
'1'	49
'2'	50
'3'	51
'4'	52
'5'	53
'6'	54
'7'	55
'8'	56
'9'	57
'a'	97
'b'	98

'c'	99
'd'	100
'e'	101
'f'	102
'g'	103
'h'	104
'i'	105
'j'	106
'k'	107
'l'	108
'm'	109
'n'	110

'o'	111
'p'	112
'q'	113
'r'	114
's'	115
't'	116
'u'	117
'v'	118
'w'	119
'x'	120
'y'	121
'z'	122

'_'	95
'`'	96
'{'	123
'|'	124
'}'	125
'~'	126
' '	32
'\t'	9
'\n'	10
'\r'	13
'\x0b'	11
'\x0c'	12

Reading a Web Page

- Use the modules urllib or urllib3.
- variable1 = urllib.urlopen('url address')
- variable2 = variable1.read() # reads the whole page

  or
- for line in variable1: # reads one line of the web page at a time
- For more complex web-page operations, you are advised to install and use the requests module.

LAB 02d

In the book, "Alice in Wonderland" find the words caterpillar and gryphon. Read the entire book into memory. Print the location of the first occurrence of each word (find method). Also, print the number of times each word occurs in the book (count method).

If you finish early, determine the location of the last occurrence of each word. There is a method that will do this for you.

Answers:

```
Word is caterpillar
The location of the first occurrence is 45,820
The location of the last occurrence is 117,881
The number of occurrences is 28
Word is gryphon
The location of the first occurrence is 104,682
The location of the last occurrence is 143,640
The number of occurrences is 55
```

Tables

One-Dimension Example

203.76
220.76
186.96
223.66
315.65
148.32
180.76
241.80
230.02
227.91
183.14
202.05
256.70
184.93

Two-Dimension Example

UK	Buchanan	7/16/2003	10248	$440.00
UK	Suyama	7/10/2003	10249	$1,863.40
USA	Peacock	7/12/2003	10250	$1,552.60
USA	Leverling	7/15/2003	10251	$654.06
USA	Peacock	7/11/2003	10252	$3,597.90
USA	Leverling	7/16/2003	10253	$1,444.80
UK	Buchanan	7/23/2003	10254	$556.62
UK	Dodsworth	7/15/2003	10255	$2,490.50
USA	Leverling	7/17/2003	10256	$517.80
USA	Peacock	7/22/2003	10257	$1,119.90
USA	Davolio	7/23/2003	10258	$1,614.88
USA	Peacock	7/25/2003	10259	$100.80
USA	Peacock	7/29/2003	10260	$1,504.65
USA	Peacock	7/30/2003	10261	$448.00
USA	Callahan	7/25/2003	10262	$584.00
UK	Dodsworth	7/31/2003	10263	$1,873.80
UK	Suyama	8/23/2003	10264	$695.62

Lists

- Python's implementation of tables/arrays.
- Can be multiple dimensions.
  - We will only deal with two at most.
- Advantage – you use one variable name to access/store multiple items.
- Lists are produced using square brackets or the list function.
- Lists can contain items that are all the same type or many different types.
- Like strings, lists can be accessed with integer indexes.
  - An index can be a literal, a variable or an expression
- Unlike strings, lists can be changed in place (mutable).

List Operations

- x = [12, 3, 124, 56, 2]
- len(x) produces 5
- x[1] is 3, x[-1] is 2, x[1:3] is [3, 124] - slicing works
- The LIST function makes a list out of any iterable.
- The RANGE function creates an iterator of integers.
  This can be turned into a list with the list function.
  - y = list(range(1, 10 ,2))
  - y is now a list - [1, 3, 5, 7, 9]
- The in operator works on lists:
  - If 9 in y: produces a True result

List Operations

- x = [] creates an empty list
- y = 5 * [0] creates a list of five zero integers
- z = [2, 3, 4] + [5, 6] is [2, 3, 4, 5, 6]
- Lists are iterables!

```
x = [1, 34, 12]
for i in x:
    print i
```

Result:

1
34
12

- x = [1, 34, 12]
- x[0] += 1
- x is now [2, 34, 12]

- Sample - dLists1.jpg

Calculating/Printing Percentages (Review)

- Let's say I have 30 temperature readings and 10 of them are over 90 degrees fahrenheit. How do I calculate that percentage?
- One Answer:
  x = 10 / 30 * 100
  print('{0:.1f}%'.format(x)) # result -> 33.3%
- Another way using % in place of f:
  x = 10 / 30 # Multiplying by 100 no longer necessary
  print('{0:.1%}'.format(x)) # result -> 33.3%
- Either way is fine. Using the % eliminates some complexity.
- Note: Using f-strings → print(f'{x:.1%}')

Lab 3a – Probability

Plan and execute the following program. Simulate the rolling of a pair of dice 100,000 times accumulating the results of each roll in a list (e.g., the number of two's, the number of three's and so on). When finished, print the percentage of times each possible roll occurred. Visually compare your results to the mathematically derived results in the adjacent table.

2	2.78%
3	5.56%
4	8.33%
5	11.11%
6	13.89%
7	16.67%
8	13.89%
9	11.11%
10	8.33%
11	5.56%
12	2.78%

How many elements will there be? How will you initialize your list? What will you use for an index?

List Methods

- In place change vs return – be careful
- In place change (void)
  - append() vs. insert() vs extend()
  - sort() vs. reverse()
  - remove()
- Produce a return (fruitful)
  - count(), index()
- Both – in-place change and a return
  - pop()
- Sample - eLists2.jpg

Changing a List

- my_list = [1, 2, 3]
- my_list.append(4) # ok
- my_list.extend([4]) # ok
- my_list.insert(0, 4) # ok
- my_list = my_list + 4 # error
- my_list = my_list + [4] # changes locations – not a good idea as you will see later
- my_list.append([4]) # inserts a list within the list
- my_list = my_list.append(4) # wipes out list

Sample - fLists3.jpg (demonstrates iterating through a list)

LAB 03b – Filter, Map Reduce

Use the range function to create a list containing the numbers:

[2, 5, 8, 11, 14, 17]

- Use the techniques you have learned so far to:
  - Create and print a new list with only the even numbers from the original list. (filter)
  - Create and print another new list containing the square of the original numbers. (map)
  - Create a result showing the sum of all the original numbers. (reduce)
- Use normal loops to accomplish each of these tasks

Answers:

```
Original list: [2, 5, 8, 11, 14, 17]
Filter [2, 8, 14]
Map [4, 25, 64, 121, 196, 289]
Reduce 57
```

List Operations

- x = [12, 3, 124, 56, 2]
- Built-in functions such as SUM, MAX and MIN can operate on a list of numbers.
  - e.g., max(x) is 124, sum(x) is 197
  - MIN and MAX can operate on non-numerics as well.
- x =['abc', 'aaa', 'zzx', 'zzz', 'AAA', 'ZZZ']
  - max(x) is 'zzz'
  - min(x) is 'AAA'

LAB 03c

Read the trees.dat file putting each valid element into a list. The file contains the height in even feet of a large sample of California coastal redwood trees. When finished, use only built-in functions (min, max, sum, len) and normal math equations to produce a report on the screen showing:

- the number of trees,
- the average height of the trees to one decimal place,
- the height of the tallest tree, and
- the height of the shortest tree.

```
Total trees - 999
Average height - 215.4
Shortest tree - 99
Tallest tree - 316
```

LAB 03d

From exercise 10-8 in the book, "Think Python":

If there are 23 students in your class, what are the chances that two of you have the same birthday? You can estimate this probability by generating random samples of 23 birthdays and checking for matches.

Write the program such that you run the exercise at least 100 times. At the conclusion of the program print the number of times duplicate dates were detected. Use the randrange function in the random module to generate numbers from 1 to 365 to simulate dates. Use the count method to determine whether there are duplicates.

Mathematically, the probability of duplicates occurring is about 50%. Your results should be in the area of 40% to 60%.

Lists – Removing Elements

- pop() if index known or last element
- remove(element) if index not known
- del operator
- del with slice notation
- del is used for many things, not just lists

Stacks

- What's a stack? (last in / first out)
- Implementing a stack
  - my_stack = [1, 2, 3]
  - my_stack.append(8) # push with append()
  - top = my_stack.pop() # pop with pop()
- What other ways can you do this operation?

Queues

- What's a queue? (first in / first out)
- Implementing a queue
  - my_queue = [1, 2, 3]
  - my_queue.insert(0,88) # add with insert()
  - last = my_queue.pop() # remove with pop()

LAB 04a

1. Implement a stack using a list data type. Pushing five elements onto the stack. Then, remove each item by using the built-in list method pop(). Print the stack at each change. (LIFO)
2. Implement a queue using a list. Insert 5 elements onto the queue using the insert() method and empty the queue using the pop() method. Print the queue at each change. (FIFO)
3. Are there alternative ways to implement either solution?

LIFO

```
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
[1, 2, 3]
[1, 2]
[1]
[]
```

FIFO

```
[1]
[2, 1]
[3, 2, 1]
[4, 3, 2, 1]
[5, 4, 3, 2, 1]
[5, 4, 3, 2]
[5, 4, 3]
[5, 4]
[5]
[]
```

Copying vs Aliasing

- Equivalent or identical ? Use id() function or *is* operator to find out.
- When identifiers point to the same value (object) in memory.
- What do x and y equal in each case? Are they the same thing?

  x = [1, 2]; y = x; y.append(3); x == y; x is y; print(x, y)
  x = [1, 2]; y = list(x); y.append(3); x == y; x is y; print(x, y)

- Make copies: help stamp out list aliasing
  - nu_list = existing_list creates an alias
  - nu_list = list(existing_list) creates a new list*
  - nu_list = existing_list[:] creates a new list*

    * This works only on one-dimensional lists.

Two-Dimensional Lists

- Lists can be many dimensions. We will deal with two.
- List 1

eggs
milk
bread

List 2

eggs	2	dozen	free range
milk	3	quart	2 percent
bread	1	loaf	whole wheat

- x =[['eggs', 2, 'dozen', 'free range'], ['milk', 3, 'quart', '2 percent'], ['bread', 1, 'loaf', 'whole wheat']]
- x[1][2] = 'quart', x[0][1] = 2, x[2][0] = 'bread'
- If I want 1 more dozen eggs, how do I add 1 to the eggs?
- See glists4.jpg in Samples for accessing a two-dimensional list.
- See hlists5.jpg in Samples for sorting a two-dimensional list.

Lab 04b

Your data contains a partial program called payday.py. It contains a pay rate variable valued at 27 dollars per hour. It also contains a two-dimensional list. Each sublist contains the day of the week and the hours worked to the nearest quarter hour. Hours worked Monday through Friday are paid at the normal rate of 27 dollars. Saturday hours are paid at time and a half (1.5 * 27) and Sunday hours are paid at double time (2 * 27).

Add the code necessary to process the data in the list and produce a report showing the total pay for the week. You should process the list using both techniques shown in sample gLists4.jpg. While indexing through the sublist is not necessary here, it will be important later that you understand how to do it.

```
Pay for the week is $1525.50
```

Creating Two-Dimensional Lists

- Initializing when you know the size of the list.
  - Note the use of a list literal in creating each sublist.
  - Be very careful not to create an alias.
  - See hlists6.jpg and hlists7.jpg in Samples.
- Initializing when the size of the list is unknown.
  - Build the sublists one at a time with append or insert.
  - Use the else option of **for** or **while** in doing so.
  - The else option will be executed when the **for** or **while** is complete.
    - The only way to skip the else is to break out of the loop.
  - How is this useful? See hlists8.jpg and hlists9.jpg in Samples.

Lab 04c

Process all of the records in the tmpprecip.dat file. Accumulate in a two-dimensional list the total precipitation by year. A partial format of the records in this file is repeated here.

Columns	Content
5 – 8	Year
9 – 13	Precipitation in the format dd.dd (inches)

When you are finished creating the list, sort it by year and print the results. A partial answer is shown here. Use the technique shown in sample hlists8.jpg

```
1900 37.19
1901 16.44
1902 24.79
1903 33.11
1904 29.38
1905 32.59
1906 20.42
1907 27.77
1908 28.52
1909 14.92
1910 16.22
1911 18.68
1912 23.73
1913 37.68
1914 33.67
1915 27.28
1916 27.66
1917 10.11
1918 29.91
1919 50.30
1920 19.56
1921 28.53
1922 24.59
1923 32.71
1924 23.65
1925 14.99
```

Lab 04d

Read the data in tmpprecip2012.dat and create a two-dimensional list containing all the data that will allow you to print a report by month of the following:

Average high temperature,

Maximum high temperature,

Minimum high temperature

Once that works, try your program on tmpprecip.dat. It contains over 100 years of daily data.

The format of the data is repeated here:

Columns	Content
1 – 2	Month
3 – 4	Day
5 – 8	Year
9 – 13	Precipitation - format dd.dd in inches
14 – 16	High Temperature (integer)

Before you write any code, be sure to answer these questions:

- **How many sublists will you need?**
- **What elements will they contain?**
- **How will you initialize them?**
- **How will you access each sublist?**
- **How will you access each element inside each sublist?**

Lab 04d

Read the data in tmpprecip2012.dat and create a two-dimensional list containing all the data that will allow you to print a report by month of the following:

Average high temperature,

Maximum high temperature,

Minimum high temperature

Once that works, try your program on tmpprecip.dat. It contains over 100 years of daily data.

The format of the data is repeated here:

Columns	Content
1 – 2	Month
3 – 4	Day
5 – 8	Year
9 – 13	Precipitation - format dd.dd (inches)
14 – 16	High Temperature (integer)

2012 report →

1	68.4	78	53
2	66.2	85	43
3	76.2	88	49
4	85.2	95	76
5	87.9	96	71
6	95.3	106	88
7	94.9	100	84
8	98.5	103	91
9	90.5	99	73
10	80.5	90	59
11	74.3	86	53
12	68.8	82	48

Things to Keep in Mind

- Most list methods return None.
  - Watch for in-place modification.
- Be very watchful for aliases.
- Lots of legitimate ways to change a list:
  - add: append(), insert(), extend()
  - delete: pop(), remove(), del[slice]

Tuples Are Sequences Too

- Tuples are basically immutable lists.
  - They are ordered and are iterable.
  - The len function works. So does the in operator.
  - Tuples can be sliced.
- Defined directly by parentheses or tuple function.

  x = (1, 2, 3)
  y = tuple(iterable)
  z = (12,) a single item tuple

  - Iterable can be any type, even another tuple
- x = () is an empty tuple.
- Tuples are typically used as return values.
- Tuples show up in lots of places.

How Are Tuples Used?

- Functions can return only a single object
  - But a tuple is an object, so for > 1 return object, use a tuple
- Try x = divmod(7, 3)
  - What data type is x? What values does it contain? Why?
  - Try x, y = divmod(7, 3) What are the data types now?
- Zip two lists together.
  - x = [1, 2, 3]; y = [4, 5, 6]; z = zip(x, y) What is z now? (Put result in a list to see the result)
  - Get used to this structure. We will see it in dictionaries.
  - Can I sort this result? How?
- Two list methods work with tuples. Which ones?

More Tuple Operations

- Sort an immutable or any iterable?
  - Use the sorted function which leaves the source unmodified.
  - try x = (1, -42, 138, 18); y = sorted(x)
  - What is the result of the above operation?
  - What data type is y?
  - z = reversed(y) What is z?
- Relational operators work as expected
  - (1, 2, 3) < (1, -2, 3) ?
  - What is the result of x = 1, 2, 3 < 1, -2, 3 ?

Collect/Unpack Arguments

- You can write a function that takes a variable number of positional arguments.
- Likewise, you can unpack a collection of objects to be passed to a function.
- Usually, you will use these capabilities in these cases:
  - You are writing a function to accept a variable number of arguments.
  - You are using a function written by someone else.
  - You are using arguments entered from command line.
- We discuss these here because most of these capabilities involve lists or tuples.

Parameter Collectors

- def fn(*varname) varname is usually args
  - The * is used only in the function definition
- *varname parameter receives all excess positional arguments
  - "Packs" them into a tuple
- There is a similar operator for keyword parameters (kwargs).
  - Good explanation here, but it gets complex.
- If you have a variable number of parameters, how do you test them for validity?
  - Use isinstance built-in function to test argument type.
  - Example: if isinstance(x, type) where type is int, float, str, etc.
  - Or: if isinstance(x, (type1, type2, …)) for multiple types
  - Do not put the type in quotes.
  - some valid types - float, int, str, list, dict, tuple, set, bool

LAB 05a

There is a program in your data file called temp_convert.py. It has a function that converts a Fahrenheit temperature to centigrade. Change this program to accept a variable number of temperatures per function call and process all of them. Print the collector argument and its type. Use the isinstance function to verify the type of each parameter as you iterate through it. Each parameter should be either int or float. Reject all others. Test with invalid data.

Example function call:

```
fahrenheit_to_centigrade(72, -10.5, 'a', 111, 55)  # function call
```

Have the function parse/test the arguments and print all results.

Unpacking a Collection

- Hypothetical situation:
  A function is expecting a variable number of positional parameters, but you have the data in a collection such as a list or tuple.
  - In the previous lab, passing a list or tuple will be received as one item.
- Using the * in front of the name in the function call unpacks the collection.
- Example:

```
def func_name(*args):
    print(len(args), args)
x = [1, 2, 3, 4]
func_name(x)  # sends one item (a list) to the function.
func_name(*x)  # sends four integers to the function
func_name(*x[1:])  # what does this do?
```

Command-Line Parameters

- You can send parameters to your program from the command line.
- This is done through the argv variable in the sys module.
- Do the following:
  - Create the following two-line program:
    from sys import argv
    print(type(argv), argv)
  - Execute your program from command line twice:
    python progname.py (In Linux or MacOS use python3)
    python progname.py 2016 04 13
  - What data type is argv? What is always the first item in argv?

LAB 05b

Create a copy of the program from Lab 05a and change the main program to accept a variable number of parameters from the command line. Send those parameters to your function which is still accepting a variable number of inputs. As before, have the function parse/test the arguments using different tools as necessary. Print all results.

Example command with parameters:

```
python lab_05.py 72 -10.5 a 111 55      #command line parameters
```

Review Lists and Tuples

- Lists are mutable tables of data in one or more dimensions.
- Tuples are immutable tables of data that show up in many places.
- Functions can accept a variable number of positional parameters.
  - The collector name (usually args) is preceded by an asterisk.
  - The data type of the collector is a tuple.
- The elements of a tuple or list can be sent to a function as individual positional parameters.
  - In the function call, place an asterisk in front of the variable name of the tuple or list.

Dictionaries

- The purpose of a dictionary is to associate a key with a value for very fast lookup.
  - Dictionaries are much more efficient in some circumstances than lists.
- Dictionaries can be created in two ways:
  - using the dict built-in function
  - using braces – x = {} is an empty dictionary.
- Keys can be any immutable type : e.g., numbers, strings, tuples.
- Keys are hashed for fast lookups. (See Python Notes for a discussion of hashing)

Dictionaries

- Examples:
  dict_01 = dict() creates an empty dictionary
  dict_02 = {'sun': 1, 'mon': 2, 'tue': 3, and so on}
  dict_03 = dict(sun = 1, mon = 2, tue = 3, and so on)
- General Information
  - Order of items unknown until Python version 3.6
    - Then dictionaries are maintained by insertion order.
  - Accessing - dict_02['mon'] returns 2
  - KeyError access exception
  - len() tells you the number of key:value pairs
  - The in operator works on keys only
  - A for loop iterates over keys only.
  - keys(), values(), and items() methods create a view of the specified parts of the dictionary.
- Review the sample - jDictionary1.jpg

Dictionaries as Counters

- Key = item to count, Value = count
- Example:

```
word = 'abracadabra'
d = dict()
for c in word:
    if c in d:
        d[c] = d[c] + 1
    else:
        d[c] = 1
print(d)
```

Result (not sorted) ➔ {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}

➢ See sort_by_count.py in Demo Programs for an example of unloading and sorting dictionaries

LAB 06a

Read the book, "Alice in Wonderland" into memory. Create a dictionary counting all the printable characters excluding whitespace. Be sure not to count upper- and lower-case letters separately. Creating the dictionary is the most important part of this lab. When done, print the top 30 most frequently occurring characters along with the number of occurrences.

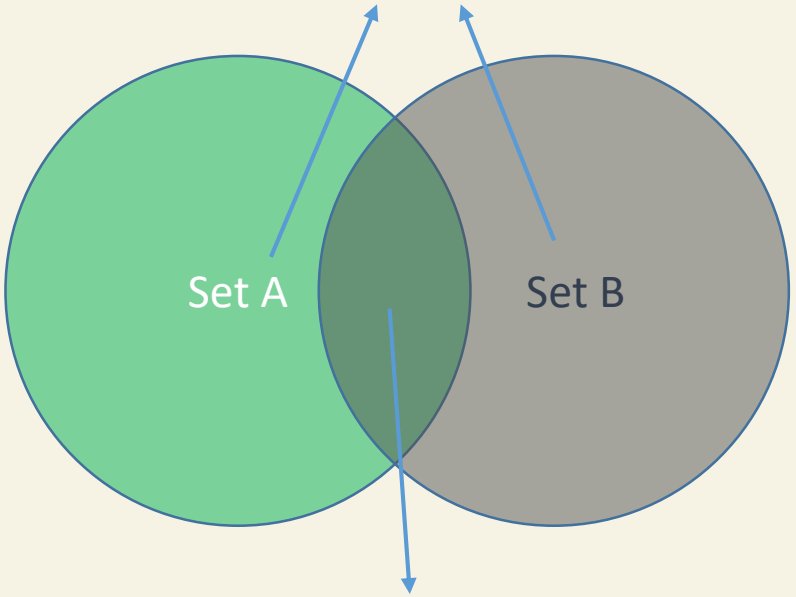
If you have time, print five character/occurrences combinations per line. Make sure all the elements of the printed lines form neat columns.

Answers:

E 13,575	T 10,689	A 8,791	O 8,146	I 7,515
H 7,375	N 7,016	S 6,500	R 5,438	D 4,931
L 4,716	U 3,468	' 2,871	W 2,676	G 2,531
, 2,418	C 2,399	Y 2,262	M 2,107	F 2,001
P 1,524	B 1,475	K 1,158	. 990	V 846
- 669	! 450	: 233	Q 209	? 202

Sets

Union Set A and Set B (No duplicates).
All unique items in both sets



Intersection Set A and Set B (No duplicates).
all items that are in both sets.

Sets

- Sets are the final data type we will study
- Sets are unordered. No indexing or slicing.
- As with all data structures, the len() built-in function is operable. Also, the **in** operator is active.
- Sets are mutable and can be changed in place.
- Each entry in a set must be unique and immutable. No duplicates.
  - Attempts to place duplicate entries in a set are simply ignored.
- x = set() creates an empty set
- x = {1, 2, 7, 81, 'ert'} creates a set with 5 entries
- x = {} creates an empty dictionary, not a set – be careful!
- y = [1, 2, 3, 2, 7, 4, 1]
- x = set(y) what does x contain?
- The set function creates a set out of any iterable.

Sets

- Do the following operations in the shell:

```
x = set()
id(x)
x.add(10); print(x)
id(x)
y = [1, 2, 3]
x.update(y); print(x)
id(x)
x.discard(10); print(x)
id(x)
x.remove(10)
```

- Now, review iset0.jpg in Samples

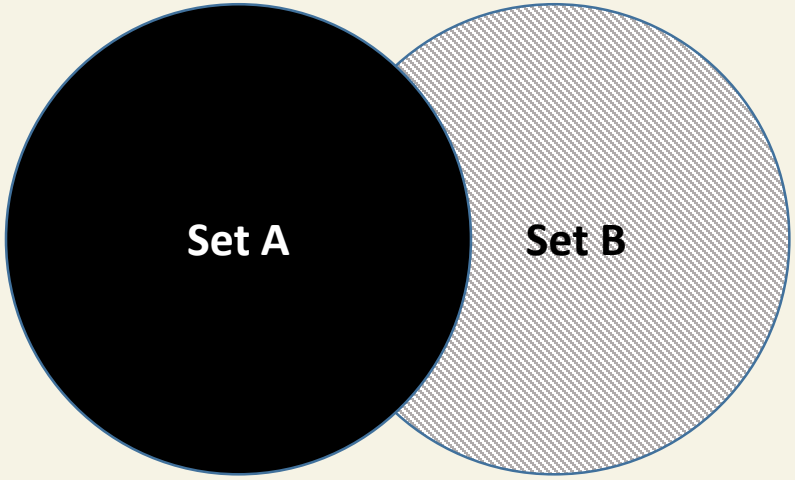
LAB 06b

Redo lab 03d in which you determined the probability of two or more people in a group of 23 had the same birthday. This time, create an empty set and place each random number you generate in the set using the add method. What happens to the duplicates? How does this help you determine if there were duplicates?

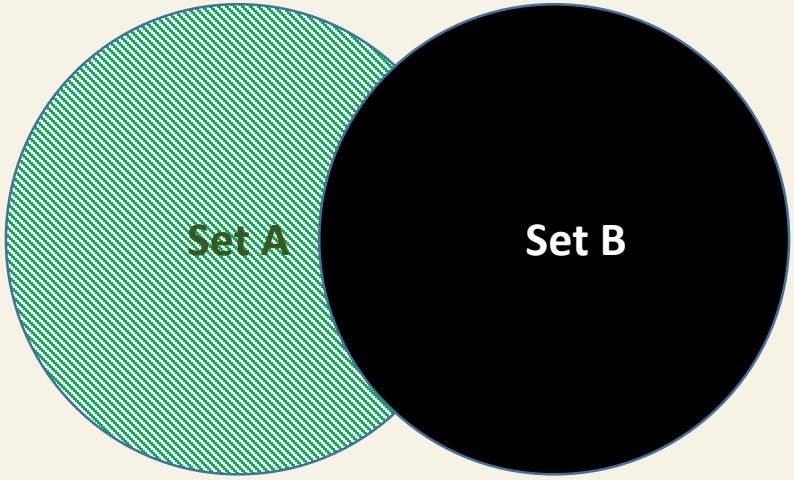
My version of lab 03d is available online.

Subtracting Sets

Set B – Set A



Set A – Set B



Sets

- Demo
  - set1 = set('himalayas') will contain h, i, m, a, l, y, s
  - set2 = set([1, 2, 3, 2, 6, 3, 1, 5]) will contain 1, 2, 3, 6, 5
  - set = {12, 2, 104, 18} creates a set in Python 2.7 and 3.x
  - union: sum of both sets with duplicates removed
  - intersection: in both sets
  - difference: removes like items in one set from another
- See the set methods in Python Notes. See samples iSets.jpg (1 & 2)

LAB 06c

In your data file is a program named servercheck.py. It reads two files (servers and updates) and converts the contents into two sets. The updates are not always correct. You will find all of the set operations/methods in Python Notes. Using just these operations/methods, your job is as follows:

1. Determine whether the list of updates exists in the master server list. Print a message indicating whether or not this is true.
2. If it is not true (and you know it isn't), create a new set containing the update items that are NOT in the master server set. Print the number and names of the unmatched servers.
3. Create a new master server set that excludes the valid updates.
4. Print the number of items in the original master server set and the new master server set as well as the number of valid updates.
5. Write the contents of the new master server set to a printable external file using the writelines file method. (See Python Notes)

LAB 06c

In your data file is a program named servercheck.py. It reads two files (servers and updates) and converts the contents into two sets. The updates are not always correct. You will find all of the set operations/methods in Python Notes. Using just these operations/methods, your job is as follows:

Answers:

```
Some updates not found in the master list
3 server updates not found
22RHLS-AS-101
22RHLS-SM-002
22RHLS-AS-100
Original servers 109
Valid updates 17
New servers 92
```

Strings (Again)

- In the last lab, we ignored the newline (\n) characters after each server name.
- We can change all of that with two string methods: splitlines and join.
- Splitlines creates a list of lines broken on line boundaries. (See Python Notes for details)
- Join concatenates all of the entries of a collection (e.g., a list). It separates them using the string upon which the method is acting:
  ex: somestring.join(somecollection)
- What does this produce?
  my_lst = 'first\nsecond\nthird\nfinal\n'.splitlines()
  prline = '\n'.join(my_lst)
- Do the above operations in the shell. Display each result.

LAB 07

Change the program you developed in the last lab to remove the newline characters from the data you read in. Then you have to put them back in the data you write. In this case, how will you read the server data? How will you write the new server file? Be sure to answer these questions before you write any code. When finished, you should be able to open the new server file with any text editor and have it display one server per line.

Strings (Again)

- split() - delimiters.
- What do these operations do?

```
linein2 = 'first:second:third:fourth:last'
line2 = linein2.split(':')
linein = "\nA serious error   has occurred on your watch\r\n"
line = linein.split()
```

- Remember, the string module has useful variables. (e.g., punctuation)
- Samples - kStringMethods2.jpg

LAB 08

Use the split() method to process the following files: gdp.txt and split.txt. Examine each file and determine how best to separate the various elements to accomplish the assigned task.

In the first file, each record has three elements: country name, total population and gross domestic product (GDP) in millions of dollars. Your job is to calculate the GDP per person for each country and print out country and GDP/person in descending order of per-capita GDP. Format the results for a professional look. The data itself comes from Wikipedia, so don't take it seriously.

In the second file, determine how many words there are in the file and how many of those words are unique. Then print out each unique word in ascending order. Be sure to change every alphabetic character to one case and remove/replace all punctuation using the replace method.

LAB 08

Use the split() method to process the following files: gdp.txt and split.txt. Examine each file and determine how best to separate the various elements to accomplish the assigned task.

Partial Answers:

Luxembourg	129,710
Iceland	92,922
Switzerland	90,532
Macau	89,665
Norway	85,204
Ireland	84,426
Qatar	70,668
Denmark	66,907
United States	65,058
Australia	63,052
Singapore	62,675
Sweden	62,548
Netherlands	58,062
Austria	57,441
San Marino	55,339
Finland	54,687

```
Words in text: 281
Unique words in text: 163
2001
2003
2008
a
adding
adopted
allows
also
amazon
an
and
as
```

Optional Lab

- Read the tmpprecip.dat file and use a dictionary to accumulate the data necessary to report the following for each year:
  - Total rainfall (precipitation)
  - Maximum high temperature
  - Minimum high temperature
  - Average high temperature

The format of the data is repeated here:

Columns	Content
1 – 2	Month
3 – 4	Day
5 – 8	Year
9 – 13	Precipitation - format dd.dd
14 – 16	High Temperature (integer)

- Use a separate dictionary to report the average precipitation by month.
- Create well-formatted reports from each dictionary
- The format of the data is repeated on the next slide along with partial answers.
- Once the program is working, exclude all data prior to 1970. Then run again excluding all data after 1970. Note the differences in rainfall totals.

Optional Lab Partial Answers

Year	Rain	Max Hi	Min Hi	Avg Hi
1900	37.2	100	42	79.6
1901	16.4	103	36	81.2
1902	24.8	103	39	80.2
1903	33.1	99	37	76.1
1904	29.4	97	40	80.2
1905	32.6	101	27	77.7
1906	20.4	102	35	78.9
1907	27.8	105	41	80.7
1908	28.5	102	42	80.2
1909	14.9	107	26	81.4
1910	16.2	104	42	81.2
1911	18.7	102	34	80.6
1912	23.7	104	29	78.4
1913	37.7	101	33	78.1
1914	33.7	102	41	77.9
1915	27.3	101	38	79.5
1916	27.7	100	37	80.9
1917	10.1	104	34	80.6
1918	29.9	104	27	79.5
1919	50.3	97	41	76.7
1920	19.6	100	36	78.8
1921	28.5	100	44	81.7
1922	24.6	100	34	79.9

01	1.65
02	1.71
03	1.83
04	2.77
05	3.80
06	3.10
07	2.24
08	2.19
09	3.42
10	3.04
11	2.02
12	1.75

Average Annual Rainfall - 29.5

Next Steps

Before going on to Python III:

- Review chapters 1 – 10 in, "Python for Everybody." Make sure you cover the vocabulary and exercises as well. This is basic foundational material. You should be reasonably comfortable with it.
- Do the optional lab if we didn't have time in class. Usually, we don't have time. It requires you to use a list as the value portion of a dictionary.
- In the book, "Think Python," complete exercise 8.5 on page 80. It is a simple encryption algorithm.
- Make sure you have done the Take-Home Lab in the slide following this one. I will send you the completed labs upon request.
- Understand all of the labs from Python II.

Next Steps

Do the following lab:

In the data from Python II find, "alice_in_wonderland.dat." You will also find a file labeled, "words.txt." The latter file contains over 100,000 English-language words. Your job is to perform the following:

Create a dictionary for counting using the entries from words.txt as the keys.

Parse the text in alice_in_wonderland.data isolating each word. This requires removing/replacing all punctuation and using the split method. Make sure the words in the book are all lower case. (Hint: replace all punctuation with spaces except the apostrophe. Replace it with a zero-length string.)

Find each word from the book in your dictionary and increase the count for that word by one.

If a word is not found in the dictionary, place it in a list with other unfound words.

When you have processed the entire book, determine the percentage of words in the dictionary that were used in the book and which word was used the most.

Remove the duplicates from the list of unfound words, sort it and print it using the same format shown on the next slide.

Don't try too hard to make this perfect. It won't happen!

Next Steps

The output from your program should look something like the following:

Words in English language dictionary - 113,810

Words in book - 26,694

Percentage of dictionary words used in the book is 2.19%

The word "the" was the most frequently used at 1,644 times

Words not in the dictionary:

30 ada alice alices alternately

ann arrum australia barrowful beauti

c canterbury carroll cartwheels chatte

cheshire christmas couldnt d delightful

didnt dinah dinahll dinahs dinn

doesnt dont dormouses duchesss e

edgar edwin elses elsie england

esq est everythings favourite footmans

france ful hadnt hasnt havent

….. and so on

Python III Class Structure

- Review (See questions in your downloaded data)
- Python version 3 changes
- Clarifying/Expanding – translate, with, sorting, copying
- Comprehensions, lambda functions, generators and recursion
- Classes, inheritance, namespaces, scoping and importing
- System-oriented modules
- Debugging tools
- Using the 2to3 converter.

The End