LING 408/508: Computational Techniques for Linguists

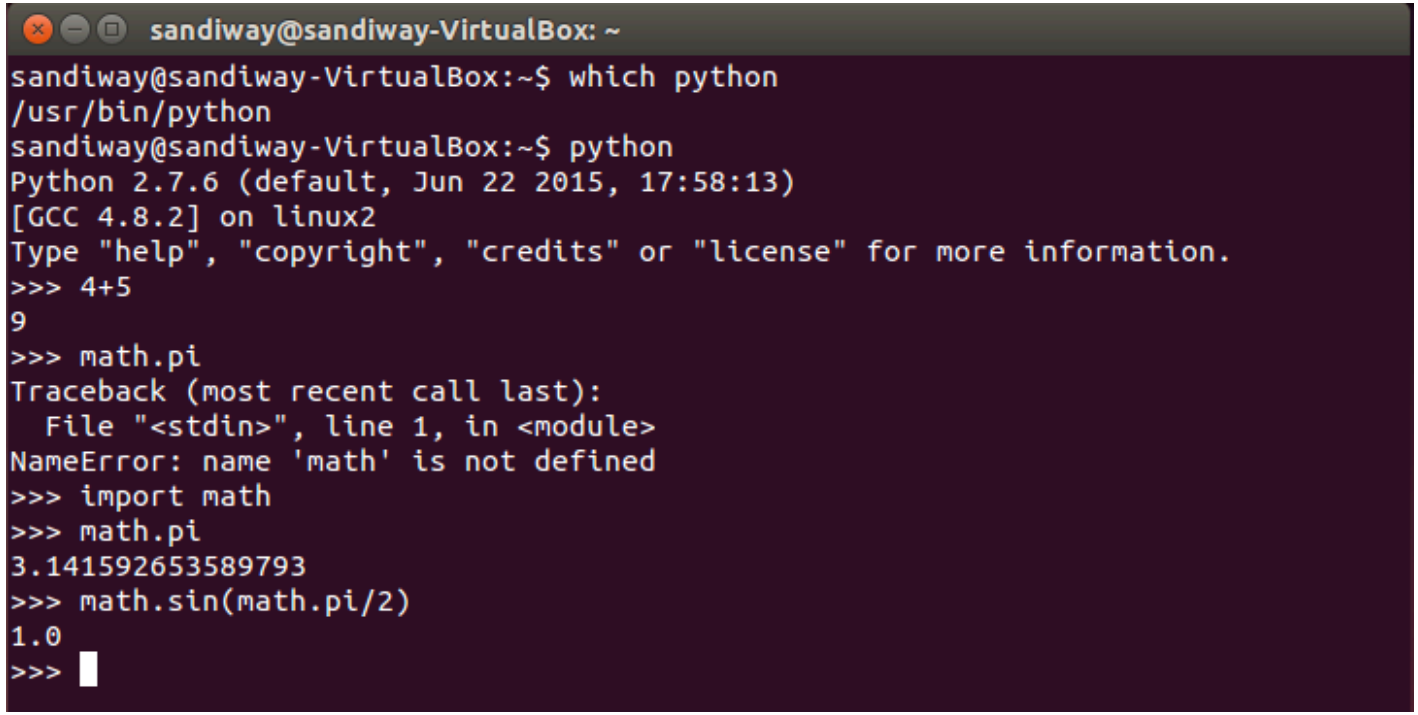
Lecture 27

Today's Topics

- Python numbers and strings
- Homework 11:
  - install nltk and nltk_data, see instructions at the end of the lecture

Python

- At the interpreter:



Python: Numbers

type: built-in function

```
>>> type(2*3-1)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type(2*3.0-1)
<class 'float'>
>>> type(2**63 - 1)
<class 'int'>
>>> type(2**1000)
<class 'int'>
>>> type("2")
<class 'str'>
>>> type("3.14")
<class 'str'>
```

arithmetic operators:

operator	operation
+	addition
−	subtraction
*	multiplication
/	division
**	exponentiation
%	remainder
`abs()`	absolute value

Table 3.1: Python built-in numeric operations.

Python: Numbers

```
import math
math.pi
```

```
>>> from math import pi, sin
>>> sin(pi/2)
1.0
```

Python	Mathematics	English
`pi`	π	An approximation of pi.
`e`	e	An approximation of e.
`sin(x)`	$\sin x$	The sine of x.
`cos(x)`	$\cos x$	The cosine of x.
`tan(x)`	$\tan x$	The tangent of x.
`asin(x)`	$\arcsin x$	The inverse of sine x.
`acos(x)`	$\arccos x$	The inverse of cosine x.
`atan(x)`	$\arctan x$	The inverse of tangent x.
`log(x)`	$\ln x$	The natural (base e) logarithm of x
`log10(x)`	$\log_{10} x$	The common (base 10) logarithm of x.
`exp(x)`	e^x	The exponential of x.
`ceil(x)`	$\lceil x \rceil$	The smallest whole number $>= x$
`floor(x)`	$\lfloor x \rfloor$	The largest whole number $<= x$

Table 3.2: Some math library functions.

Python: complex numbers

- Complex number library:
  - `import cmath`
  - https://docs.python.org/3.0/library/cmath.html

  sqrt is the square root function, e.g. `sqrt(4)=2, sqrt(9)=3` etc.

```
>>> math.sqrt(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

- *i* is *j* in Python:

```
>>> cmath.sqrt(-1)
1j
>>> i = cmath.sqrt(-1)
>>> i*i
(-1+0j)
```

Python: Numbers

```
~$ python
Python 2.7.10 (default, Oct  6 2017, 22:29:07)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 9/5
1
>>> ^D
~$ python3
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 9/5
1.8
>>>
```

- Python:
  - Number data types: int, float
  - Matters for things like division: /
- Javascript:
  - everything is a floating point number

Python

- **`range()`**: produces a list (sequence)
  - **`range(n)`** [0,1,..,n-1]
  - **`range(start,n)`** [start,start+1,..,n-1]
  - **`range(start,n,step)`** [start,start+step,…,last]
  - last=start+k*step < n
  - **`range(n,stop,-step)`** *counts down*

```
>>> for i in [0,1,2,3]:
        print i

0
1
2
3
```

(pythonbook page 40)

```
>>> for odd in [1, 3, 5, 7, 9]:
        print odd * odd

1
9
25
49
81
```

Python Program: using range()

Python 2:
use **raw_input()** instead of **input() for strings**
eval(raw_input()))is equivalent to **input()**

```
1# C-style program in Python 3
2def main():
3    print("This program calculates the future value of a 10 year investment.")
4    principal = float(input("Enter initial principal: "))
5    apr = float(input("Enter annual interest rate: "))
6
7    for year in range(10):
8        principal = principal * (1 + apr)
9
10    print("Value in 10 years is: ", principal)
11
12main()
```

convert string to float

```
[ling508-20$ python3 futval.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 2020
Enter annual interest rate: 0.035
Value in 10 years is:  2849.409496454664
[ling508-20$ python3 futval.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 2020
Enter annual interest rate: 0.04
Value in 10 years is:  2990.093455535055
ling508-20$
```

how about to 2 decimal places?

Formatted Output

- Lots of ways: (Old way in Python)

```
>>> print "x is %.2f" % 5
x is 5.00
```

```
>>> print "x is %.2f and %2d" % (5,5)
x is 5.00 and  5
```

Notation comes originally from C's `printf` function

- `printf` also is a Bash shell command

http://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm

Conversion	Meaning	Notes
`'d'`	Signed integer decimal.	
`'i'`	Signed integer decimal.	
`'o'`	Signed octal value.	(1)
`'u'`	Obsolete type – it is identical to `'d'`.	(7)
`'x'`	Signed hexadecimal (lowercase).	(2)
`'X'`	Signed hexadecimal (uppercase).	(2)
`'e'`	Floating point exponential format (lowercase).	(3)
`'E'`	Floating point exponential format (uppercase).	(3)
`'f'`	Floating point decimal format.	(3)
`'F'`	Floating point decimal format.	(3)
`'g'`	Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.	(4)
`'G'`	Floating point format. Uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.	(4)
`'c'`	Single character (accepts integer or single character string).	
`'r'`	String (converts any Python object using *repr()*).	(5)
`'s'`	String (converts any Python object using `str()`).	(6)
`'%'`	No argument is converted, results in a `'%'` character in the result.	

Formatted Output

```
<template-string> % (<values>)
```

- **%<width>.<precision><type>**
- **<type>** = d, f, s
- **<width>** = minimum number of characters; right-justified by default, -width => left-justified 0 = as wide as needed
- **<precision>** = number of places after decimal point

- e.g. **02d** two digits wide, pad with 0 if needed

Formatted Output

- **Newer way**:
  - https://docs.python.org/2/tutorial/inputoutput.html#fancier-output-formatting
  - Use {} for each argument
    - (can be numbered, e.g. {0}, {1},… or referenced by keyword {x})

```
>>> import math
>>> print 'The value of PI is approximately {0:.3f}.'.format(math.pi)
The value of PI is approximately 3.142.
```

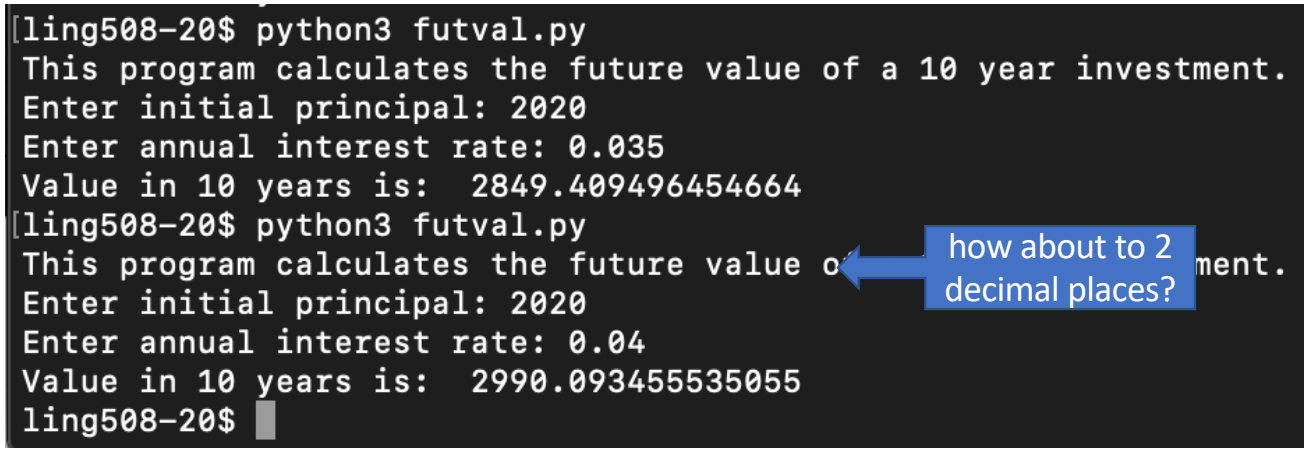
```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print '{0:10} ==> {1:10d}'.format(name, phone)
...
Jack       ==>       4098
Dcab       ==>       7678
Sjoerd     ==>       4127
```

{:**width**}

Note: use parentheses for print in Python3

Formatted Output

- Let's modify futval.py:



Python integers

- Python 2 automatically converts to type **long int** (64 bit; 2's complement) from **int** (32 bit; 2's complement)
- Python 3: int is basically **long int**, but can go to any size (*limited by available memory*):

```
>>> import sys
>>> sys.int_info
sys.int_info(bits_per_digit=30, sizeof_digit=4)
>>> sys.maxsize
9223372036854775807
>>> 2**63 - 1
9223372036854775807
>>>
```

```
>>> 2**1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729
17553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418215304647498358194
12673987675591655439460770629145711964776865421676604298316526243868372056680693 76
```

Python numbers

- explicit type coercion:
  1. `float()`
  2. `int()`
  3. `long()`
  4. `complex(`*`real`*`,`*`imaginary`*`)`
     numbers
  5. `complex(`*`string`*`)`

`not in Python 3`

complex numbers out of two floating point

String slicing

- String is like an array of characters (strings):
  - **`str[i]`** index i (from 0 to len(str)-1)
  - **`str[-i]`** index i from the end (1 = last)
  - **`str[i:j]`** slice from index i until index j-1
  - **`str[:j]`** slice from index 0 until index j-1
  - **`str[i:]`** slice from index i until end of the string

Operator	Meaning
+	Concatenation
*	Repetition
<string>[ ]	Indexing
<string>[:]	Slicing
len(<string>)	Length
for <var> in <string>	Iteration through characters

Table 4.1: Python string operations.

List vs. Strings

- Although Strings are like Lists, Lists are mutable, Strings are not.

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = 'z'
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Strings and Unicode

- ASCII ⇔ string:

```
>>> ord("a")
97
>>> ord("A")
65
>>> chr(97)
'a'
>>> chr(90)
'Z'
```

Python 3

```
>>> x = "é"
>>> x[0]
'é'
>>> x[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> ord(x)
233
>>> ord(x[0])
233
>>>
```

```
>>> x = "é"
>>> x[0]
'\xc3'
>>> ord(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
>>> ord(x[0])
195
>>> ord(x[1])
169
```

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

Strings and Unicode

- Example: u (unicode) prefix (for Python 2.7)

```
>>> x = "á"
>>> type(x)
<type 'str'>
>>> x = u"á"
>>> type(x)
<type 'unicode'>
>>> len(x)
1
>>> x = "á"
>>> len(x)
2
```

```
>>> x = u"á".encode('utf-8')
>>> type(x)
<type 'str'>
>>> x[0]
'\xc3'
>>> x[1]
'\xa1'
>>> x = "á".encode('utf-8')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3
not in range(128)
>>> x = u"á".encode('utf-8')
>>> y = x.decode('utf-8')
>>> type(y)
<type 'unicode'>
>>> y[0]
```

```
>>> x = unicode("abc")
>>> type(x)
<type 'unicode'>
```

New in version 3.3: Support for the unicode legacy literal (`u'value'`) was reintroduced to simplify the maintenance of dual Python 2.x and 3.x codebases. See **PEP 414** for more information.

Python sys.argv

- List of arguments from the command line: what's `argv[0]` then?
- can use `len()` to calculate number of arguments

```
1 from sys import argv
2 print(argv[0])
```

test.py

```
$ python3 test.py
test.py
$ python3 test.py 121 2
test.py
```

argv[1] and argv[2] ignored

Formatted Output

- Let's modify futval.py to accept arguments on the command line or via prompts:

```
[ling508-20$ python3 futval3.py 2020
This program calculates the future value of a 10 year investment.
Enter annual interest rate: 0.035
Value in 10 years is: 2849.41
[ling508-20$ python3 futval3.py 2020 0.035
This program calculates the future value of a 10 year investment.
Value in 10 years is: 2849.41
```

```
[ling508-20$ python3 futval3.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 2020
Enter annual interest rate: 0.035
Value in 10 years is: 2849.41
```

Python: Files

- Like all other programming languages, uses a file handle, called **file variable**: open()
- `infile = open("file.txt","r")` `outfile = open("results.txt,"w")`

`<filevar>.read()` Returns the entire remaining contents of the file as a single (potentially large, multi-line) string.

`<filevar>.readline()` Returns the next line of the file. That is all text up to *and including* the next newline character.

`<filevar>.readlines()` Returns a list of the remaining lines in the file. Each list item is a single line including the newline character at the end.

```
f = open(fname)
f = open(fname, encoding="utf-8")
f = open(fname, encoding="latin-1")
f = open(fname, encoding="ascii")
```

Removing the newline:

```
.strip()
.rstrip()
.lstrip()
```

```
infile = open(someFile, 'r')
for line in infile.readlines():
    # process the line here
infile.close()
```

```
infile = open(someFile, 'r')
for line in infile:
    # process the line here
infile.close()
```

Python 2.7

```
f = codecs.open('file.txt', encoding='utf-8')
```

www.nltk.org

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

SEARCH

Go

Platforms

- Today I'll go through installation for:
  1. MacOS (and Linux)
  2. Windows 10
- Your homework assignment:
  - install nltk, and
  - check to see if it works (by next time)
- In the remaining few lectures:
  - http://www.nltk.org/book/

NLTK 3.2.5 Install

- See http://www.nltk.org/install.html
- Use `pip3` (for python3) to install packages from the Python Package Index (PyPI)
- `sudo pip3 install –U nltk`

```
Collecting nltk
  Downloading nltk-3.2.5.tar.gz (1.2MB)
    100% |████████████████████████████████| 1.2MB 797kB/s
Collecting six (from nltk)
  Downloading six-1.11.0-py2.py3-none-any.whl
Installing collected packages: six, nltk
  Found existing installation: six 1.10.0
    Uninstalling six-1.10.0:
      Successfully uninstalled six-1.10.0
  Found existing installation: nltk 3.2.4
    Uninstalling nltk-3.2.4:
      Successfully uninstalled nltk-3.2.4
  Running setup.py install for nltk ... done
Successfully installed nltk-3.2.5 six-1.11.0
```

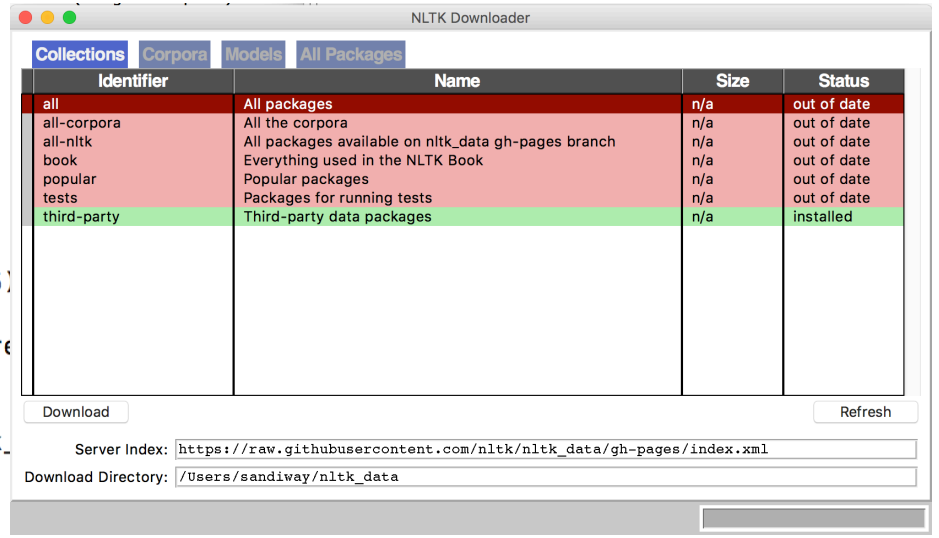
NLTK Data Install

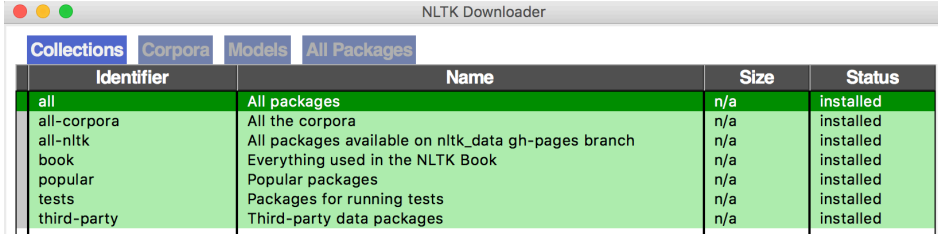
- See http://www.nltk.org/data.html
- python3

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
>>> import nltk
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_
```

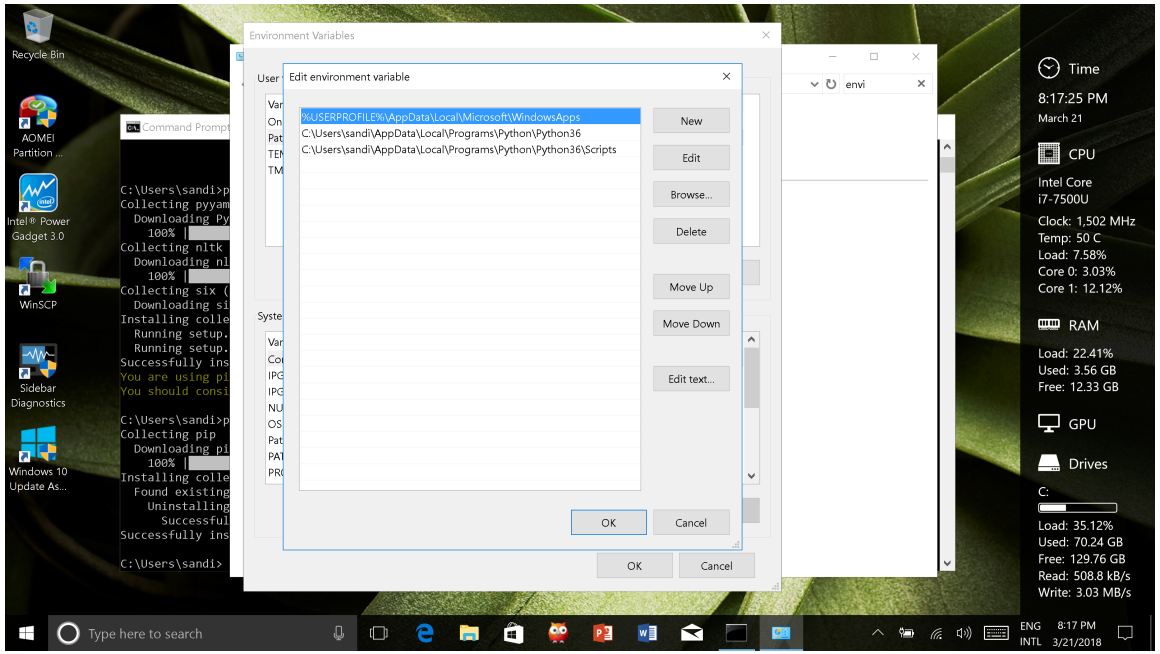
- If you get an SSL certificate error message, run:

```
/Applications/Python 3.6/Install Certificates.command
```



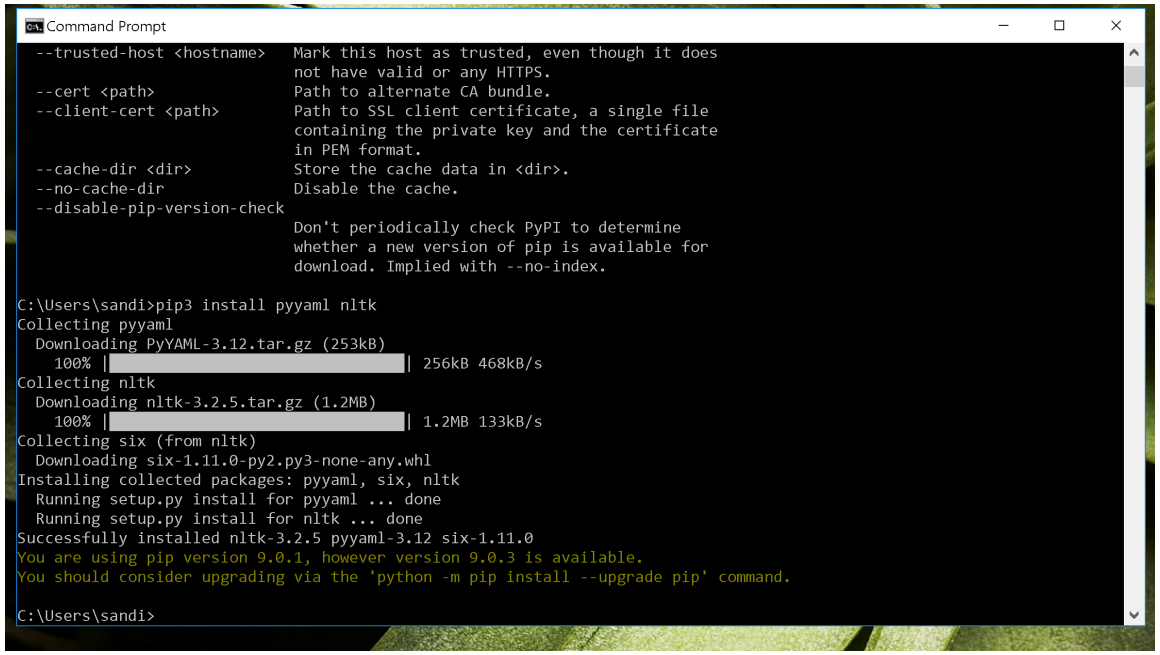


Windows 10: setup



- Environment variable PATH should be set correctly to point to Python 3 install directory
- Type in search:
  - Edit environment variables for your account

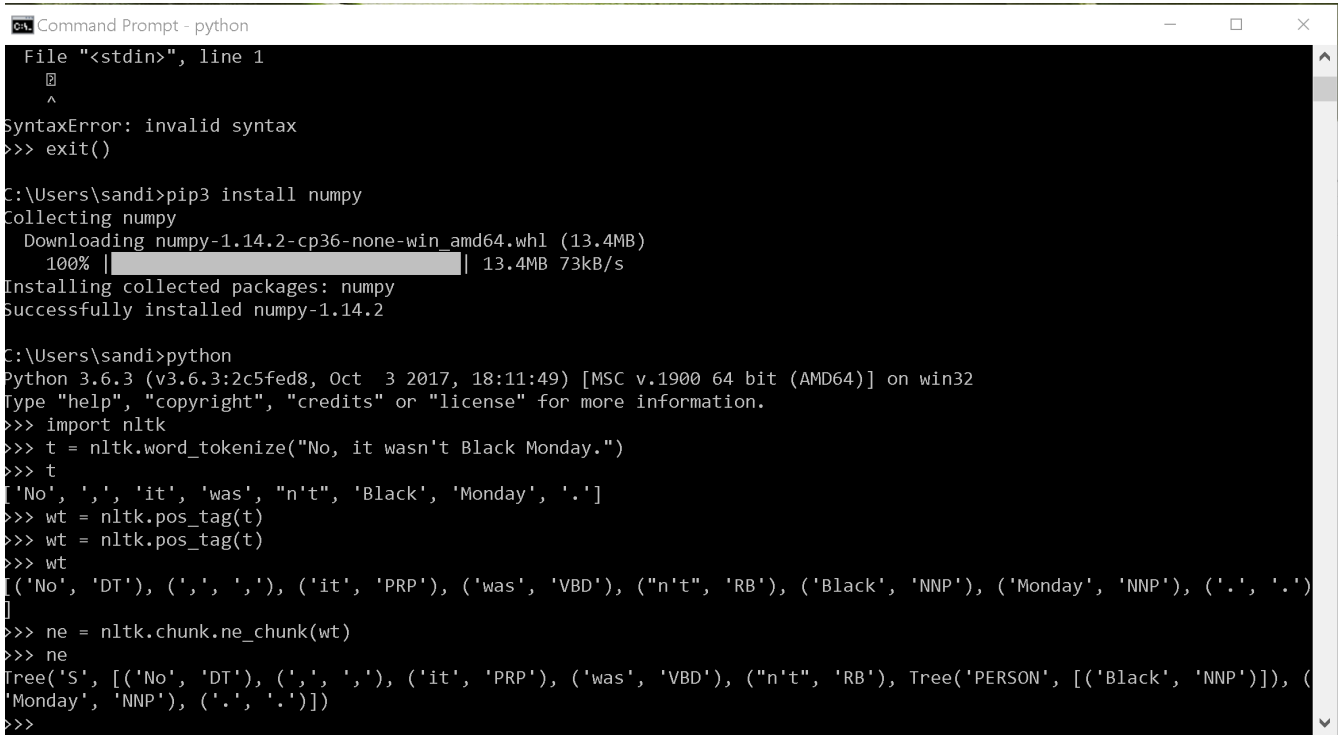
Windows 10: install nltk



- On the command line:
  - `pip3 install pyyaml nltk`
  - Package pyyaml must be used somewhere in nltk
    …

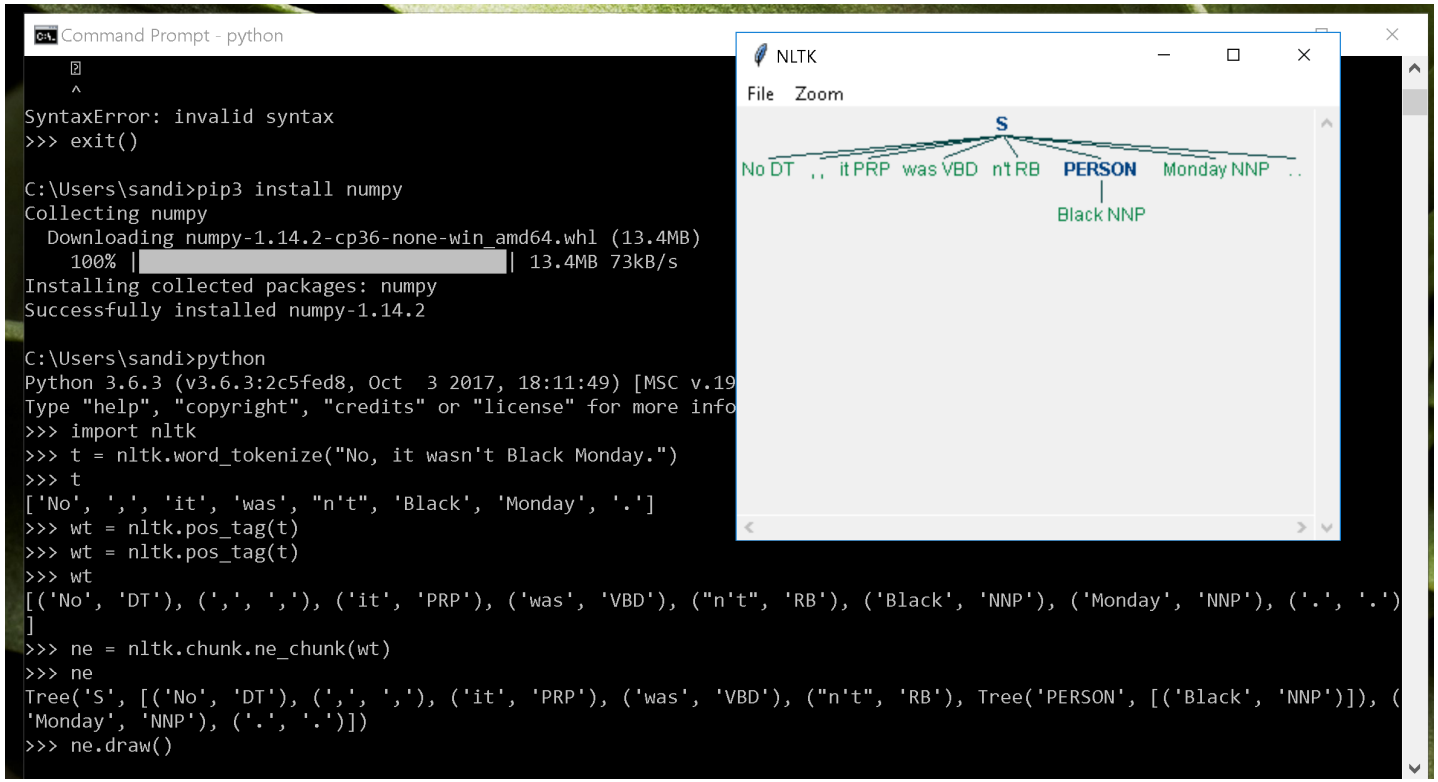
- Source: http://www.pitt.edu/~naraehan/python2/faq.html

Windows 10: install numpy and test nltk



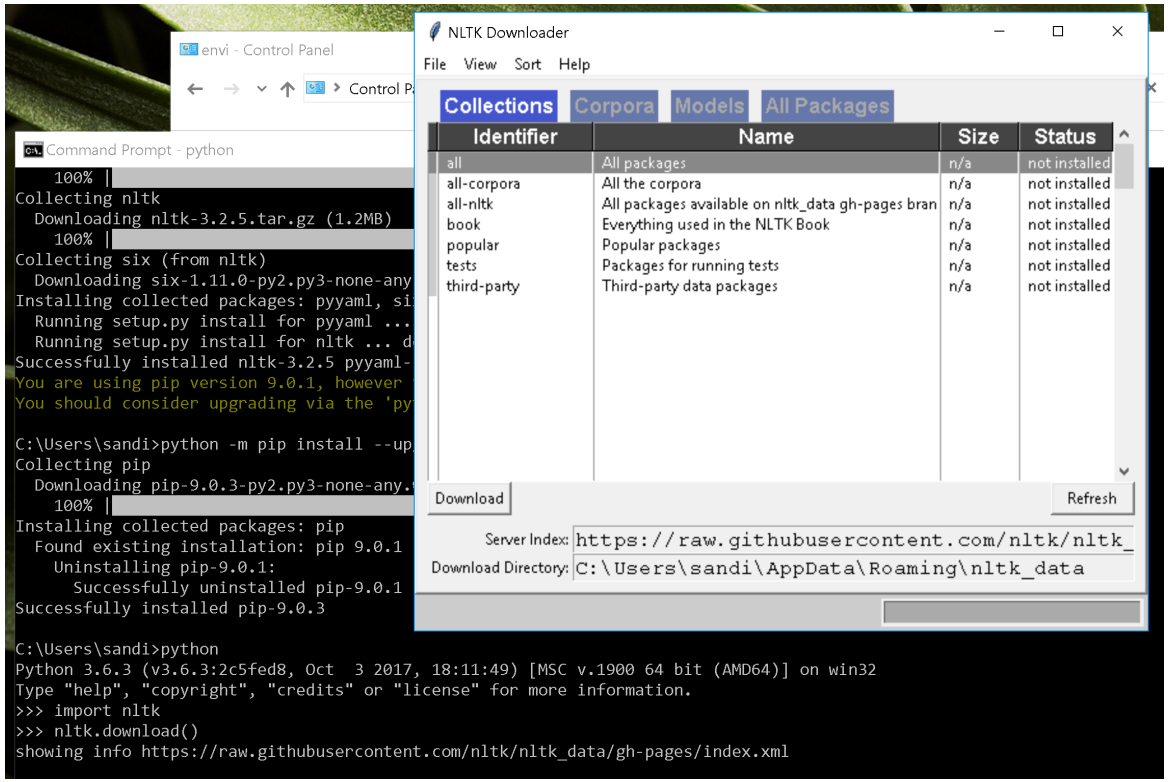
- On the command line:
  - `pip3 install numpy`
  - (the chunking algorithm uses it)
- Let's test nltk:
  - `.word_tokenize()` converts a string into words
  - `.pos_tag()` does part-of-speech tagging
  - `.ne_chunk()` does named entity recognition

Windows 10: test nltk



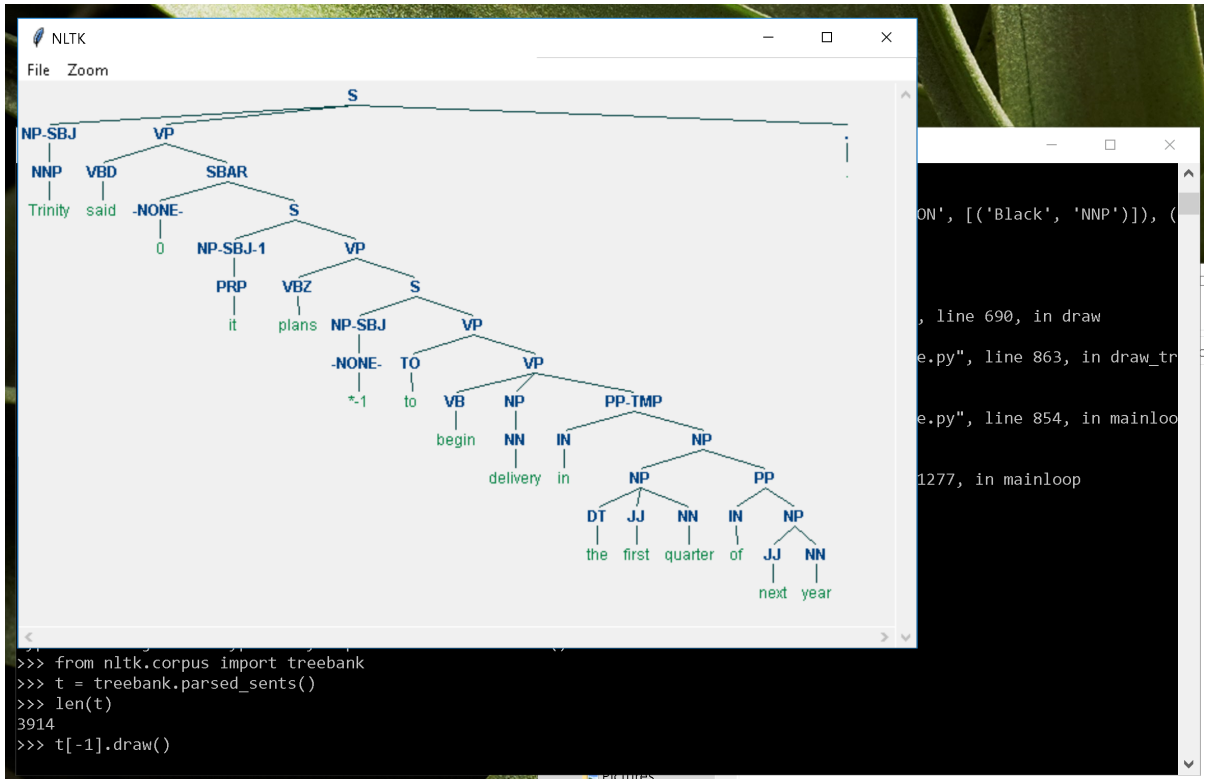
- `.draw()` takes a Tree object and draws it in a pop-up window

Windows 10: install nltk data



- Install corpus data (from inside Python) using
  - `nltk.download()`

Windows 10: test nltk data



- There is a *sample* of the well-known Penn Treebank Wall Street Journal (WSJ) corpus included
  - 3,914 parsed sentences
  - 49,000+ parsed sentences in the full corpus