Lecture 22

408/508 *Computational Techniques for Linguists*

Today's Topics

- Homework 9 okay?
  - That was installing Python 3
- Homework 10 today:
  - install `nltk` (natural language toolkit)
  - see `nltk.org` install

Today's Topics

- More on Python:
  - `range()`
    - example of use in calculating compound interest
  - `print('format string'.format())`
    - formatted printing
  - type coercion:
    - converting numbers and strings
  - `def`
    - defining functions
  - `strings and lists`
  - `sys.arg`
    - command line argument vector (ARGV)

Python: `range()`

- `range()`: equivalent to producing a list of numbers (*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 to stop*

note: excludes n!

range(Start, End, Step) (integers only!)

```
>>> range(1,10,2)
range(1, 10, 2)
>>> list(range(1,10,2))
[1, 3, 5, 7, 9]
```

note: list() converts sequence into a list

Python: `range()`

```
python
>>> for odd in range(1,9,2):
...     print(odd*odd)
...
1
9
25
49
>>> for odd in range(9,1,2):
...     print(odd*odd)
...
81
49
25
9
>>> for odd in range(9,1,-2):
...     print(odd*odd)
...
>>>
```

note: hit ENTER

note: 81 not printed!

note: 1 not printed!

Python Program: using range()

- File: `futval.py`

```
1print("This program calculates the future value of a 10 year investment.")
2principal = float(input("Enter initial principal: "))
3apr = float(input("Enter annual interest rate, e.g. 0.03 (3%): "))
4
5for year in range(10):
6    principal = principal * (1 + apr)
7
8print("Value in 10 years is: ", principal)
```

float() converts string to a floating point number

```
(base) ling508-22$ python futval.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 1000
Enter annual interest rate, e.g. 0.03 (3%): 0.05
Value in 10 years is:  1628.8946267774422
(base) ling508-22$
```

can use int() to convert to dollars. How about to 2 decimal places?

Python numbers

- explicit type coercion:
  1. `float()`
  2. `int()`
  3. `long()`
     - 64 bit integer: *not in Python 3*
  4. `complex(`*`real`*`,`*`imaginary`*`)`
     - a *complex number out of two floating point numbers*
  5. `complex(`*`string`*`)`
     - e.g. `complex('0+1j')`
  6. `str(`*`number`*`)`

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`:

```
8print("Value in 10 years is: {:.2f}".format(principal))¶
```

```
$ python futval2.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 1000
Enter annual interest rate, e.g. 0.03 (3%): 0.05
Value in 10 years is: 1628.89
$
```

function my_range

- `def` defines a function
- `my_range` uses `yield` to compute a general range on the fly.

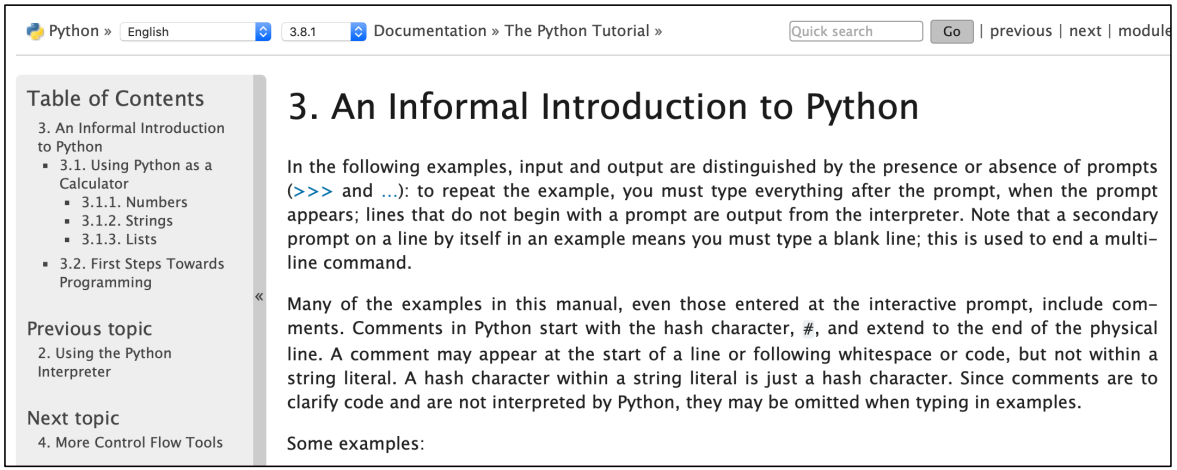
```
>>> def my_range(start, end, step):
...     while start <= end:
...             yield start
...             start += step
...
>>> my_range(0,9.9,0.3)
<generator object my_range at 0x103533580>
>>> list(my_range(0,9.9,0.3))
[0, 0.3, 0.6, 0.8999999999999999, 1.2, 1.5, 1.8, 2.1, 2.4,
2.6999999999999997, 2.9999999999999996, 3.2999999999999994,
3.599999999999999, 3.899999999999999, 4.199999999999999,
4.499999999999999, 4.799999999999999, 5.099999999999999,
5.399999999999999, 5.699999999999998, 5.999999999999998,
6.299999999999998, 6.599999999999998, 6.899999999999998,
7.1999999999999975, 7.499999999999997, 7.799999999999997,
8.099999999999998, 8.399999999999999, 8.7, 9.0, 9.3,
9.600000000000001]
```

use TAB key to indent

hit RETURN here

Python

- https://docs.python.org/3/tutorial/introduction.html
- Numbers
- **Strings**
- Lists
- Dictionaries



Python: Strings

3.1.2. Strings ¶

Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes (`'...'`) or double quotes (`"..."`) with the same result [2]. `\` can be used to escape quotes:

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing

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
```

changed!

not mutable!

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:

```
$ python futval3.py 1000 0.05
This program calculates the future value of a 10 year investment.
Value in 10 years is: 1628.89
$ python futval3.py 1000
This program calculates the future value of a 10 year investment.
Enter annual interest rate: 0.06
Value in 10 years is: 1790.85
$ python futval3.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 1000
Enter annual interest rate: 0.07
Value in 10 years is: 1967.15
```

Homework 10

- No need to submit
- But if you run into trouble, email me!

https://www.nltk.org/install.html

nltk.org

Installing NLTK

NLTK requires Python versions 3.7, 3.8, 3.9, 3.10 or 3.11.

For Windows users, it is strongly recommended that you go through this guide to install Python 3 successfully **https://docs.python-guide.org/starting/install3/win/#install3-windows**

Setting up a Python Environment (Mac/Unix/Windows)

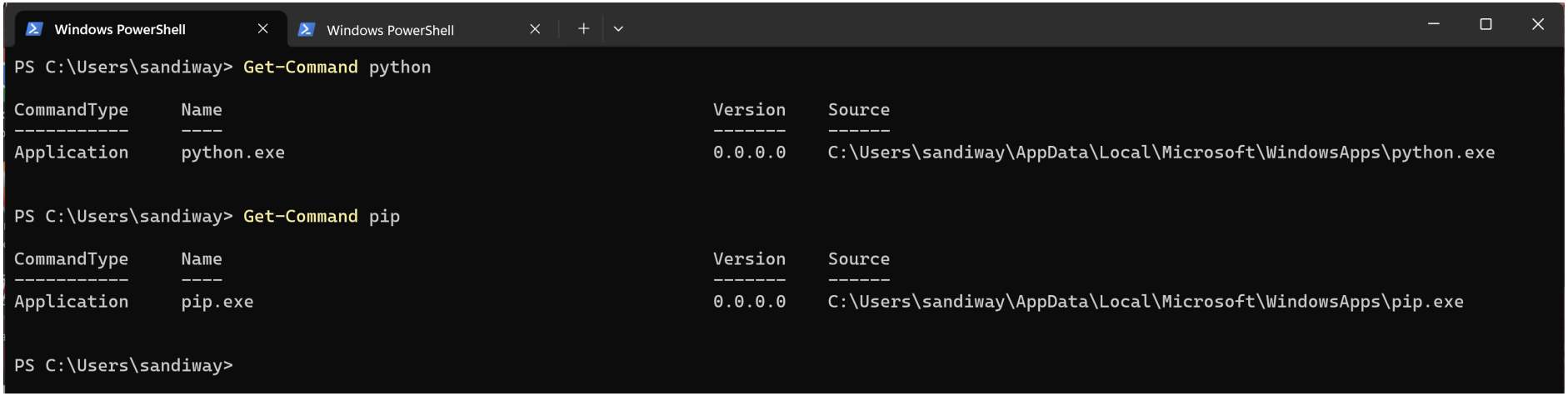
Please go through this guide to learn how to manage your virtual environment managers before you install NLTK, **https://docs.python-guide.org/dev/virtualenvs/**

Alternatively, you can use the Anaconda distribution installer that comes "batteries included" **https://www.anaconda.com/distribution/**

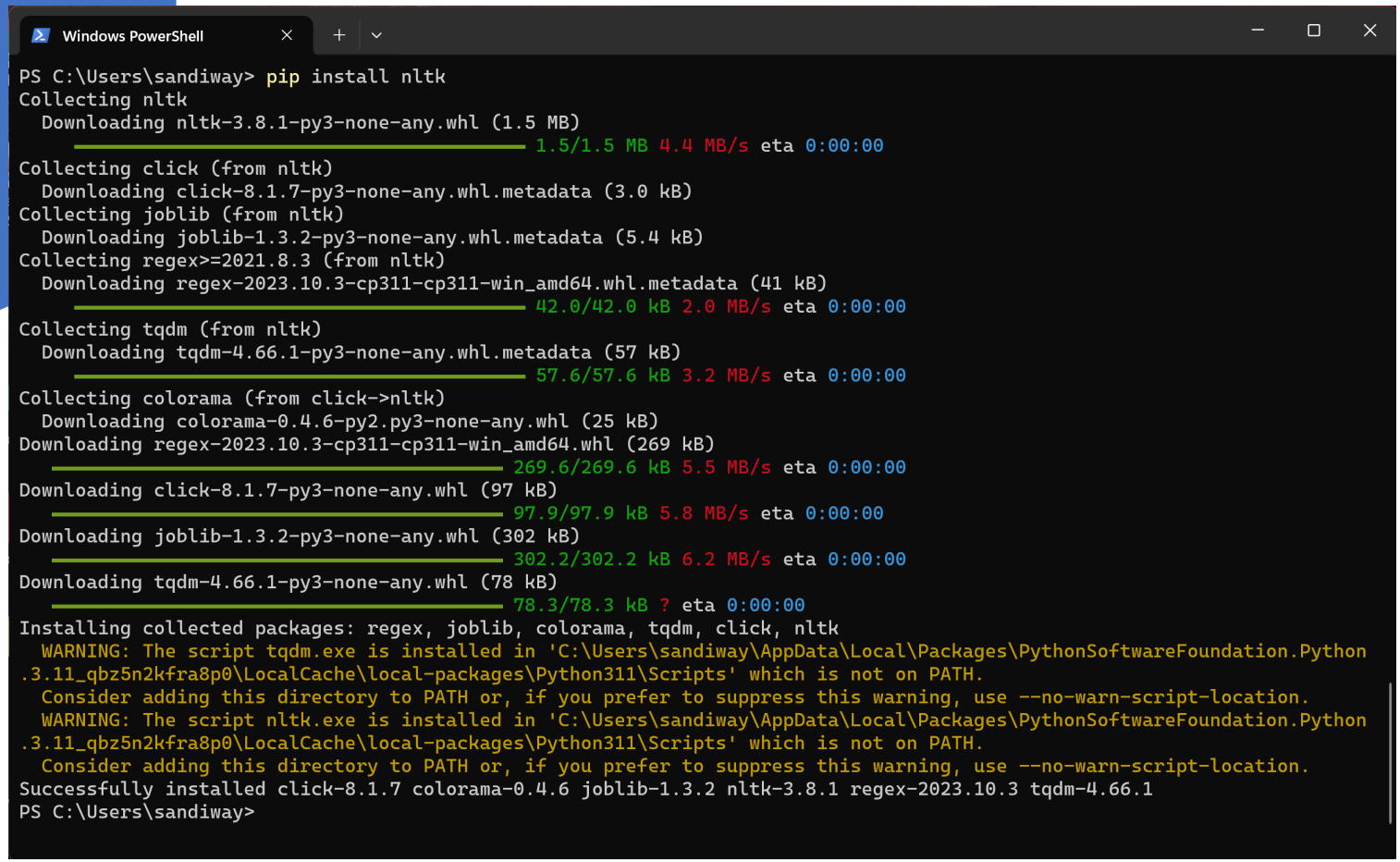
Mac/Unix

1. Install NLTK: run `pip install --user -U nltk`
2. Install Numpy (optional): run `pip install --user -U numpy`
3. Test installation: run `python` then type `import nltk`

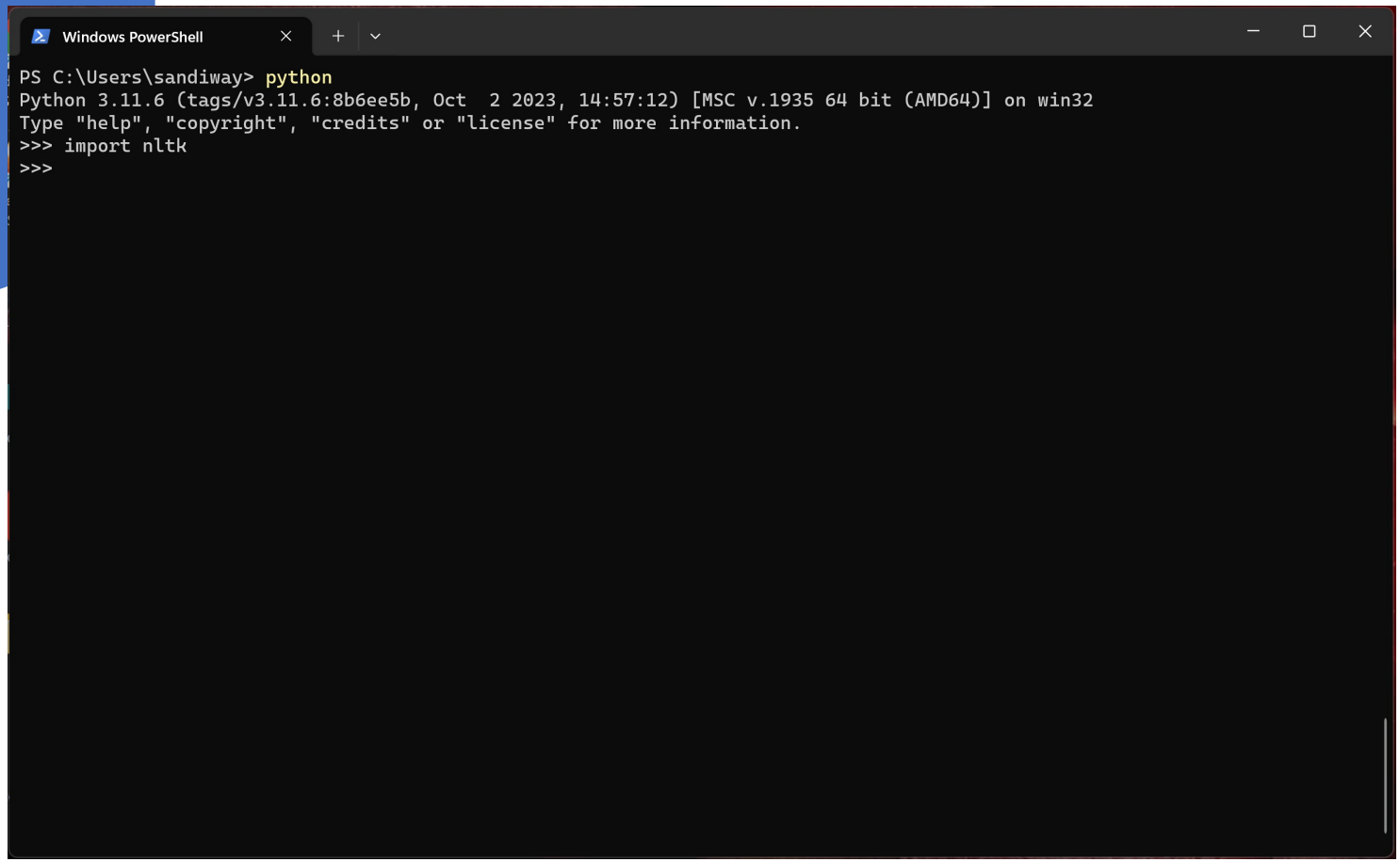
Get-Command


Windows PowerShell
Windows PowerShell
PS C:\Users\sandiway> Get-Command python
CommandType Name Version Source
Application python.exe 0.0.0.0 C:\Users\sandiway\AppData\Local\Microsoft\WindowsApps\python.exe
PS C:\Users\sandiway> Get-Command pip
CommandType Name Version Source
Application pip.exe 0.0.0.0 C:\Users\sandiway\AppData\Local\Microsoft\WindowsApps\pip.exe
PS C:\Users\sandiway>

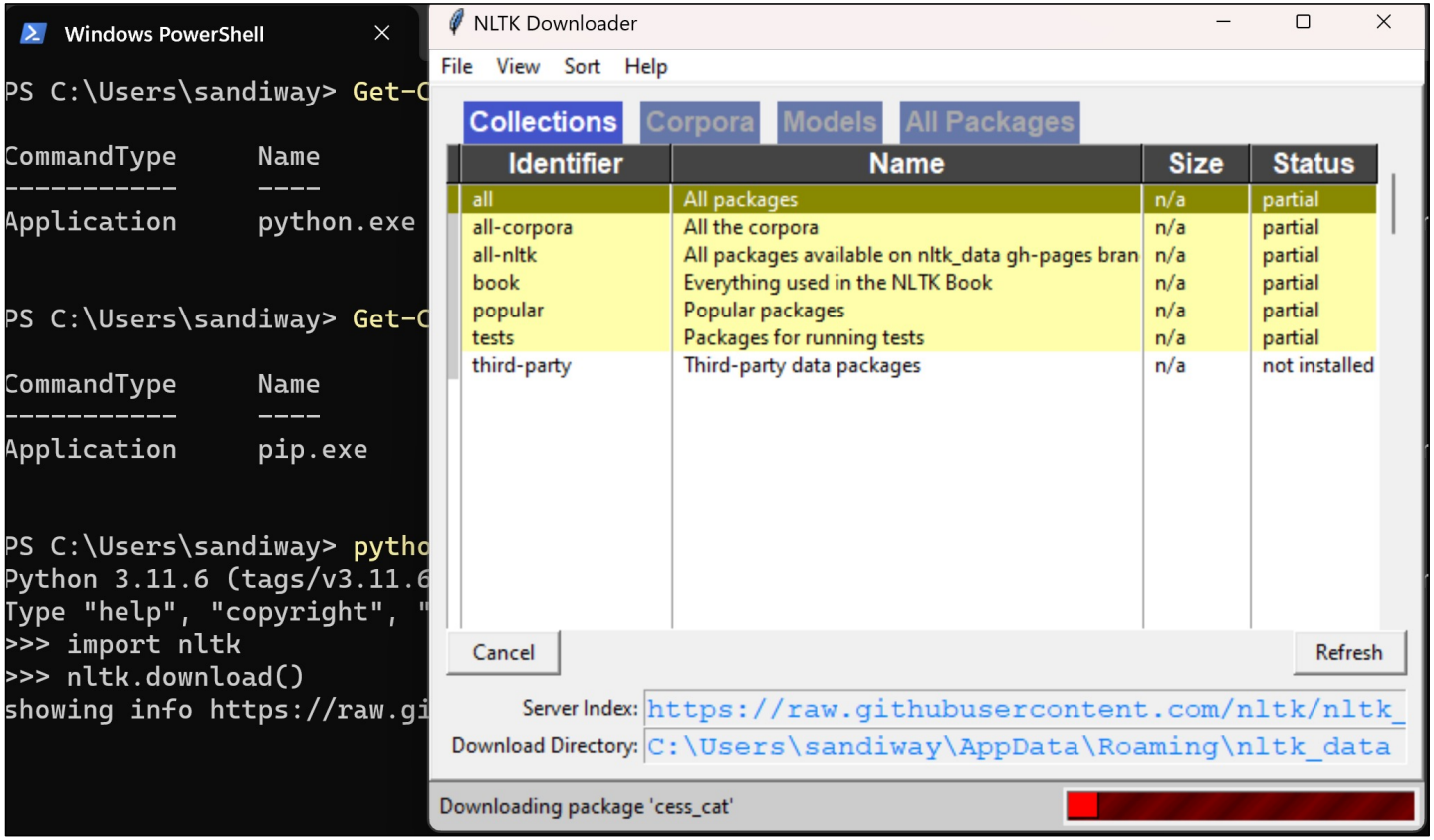
pip on Windows



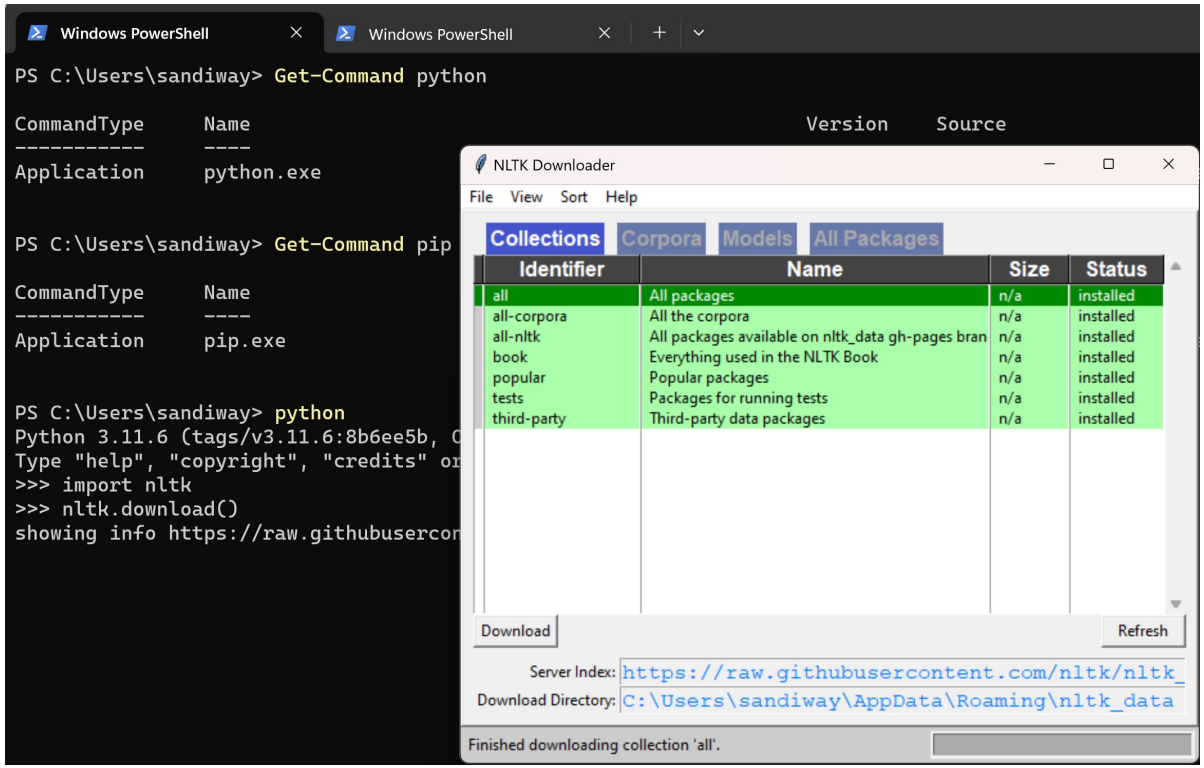
nltk on Windows

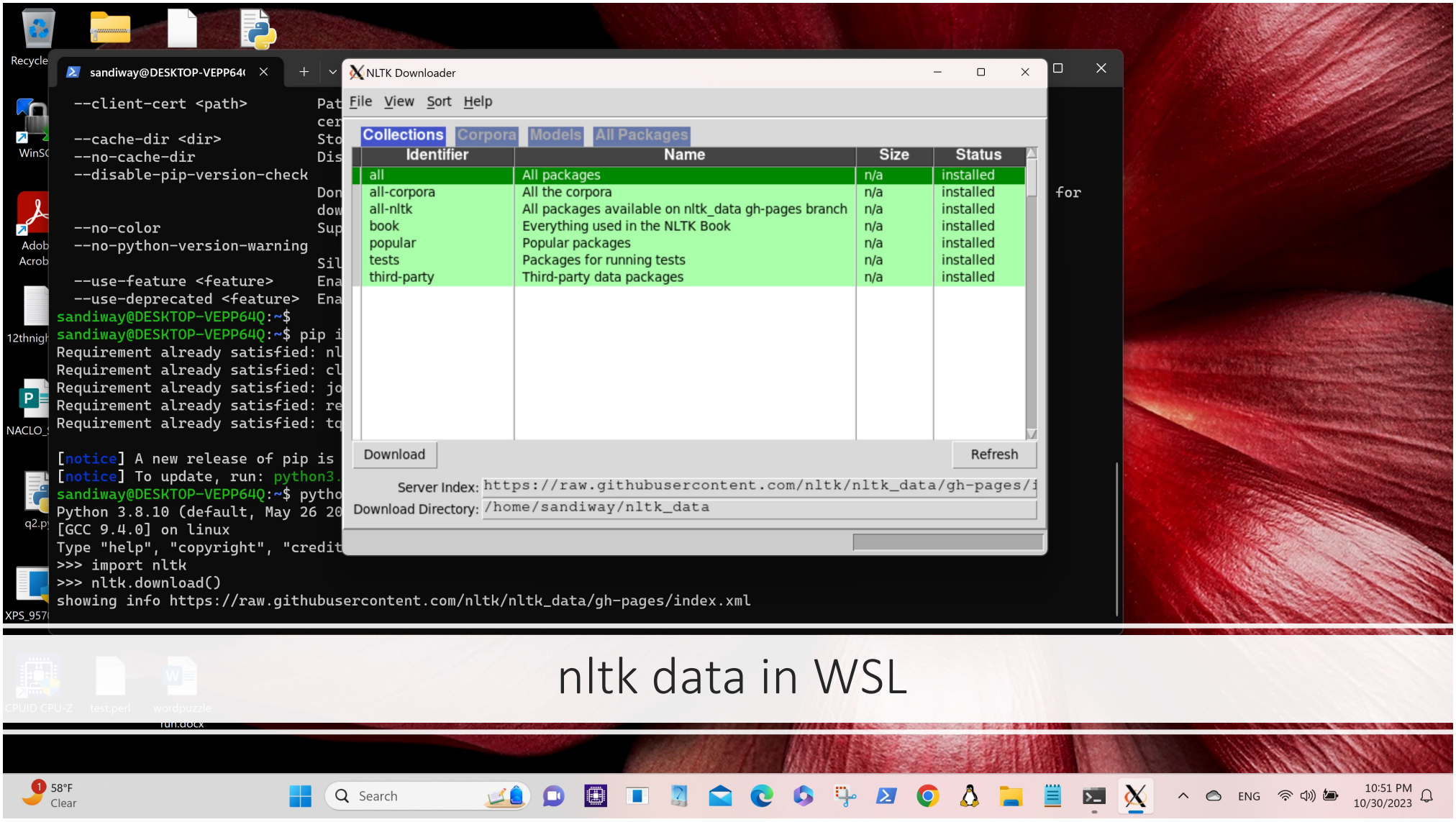


install nltk data
(select all)



nltk data installed





nltk data in WSL