Welcome to CS 134!

Introduction to Computer Science

Shikha Singh & Iris Howley

-Midterm Review Session-

```
(___)
(o o)_____/
 @@ `     \
  \ ____, /
  //    //
 ^^    ^^
```

Spring 2020

ANNOUNCEMENTS

Please read email from Shikha at 1:30pm today/Wednesday
“Midterm postponed and logistics on going remote"

- As classes have been canceled next week…

- Midterm exam has been postponed until after spring break
- TA Student Help Hours are canceled Wednesday & Thursday
- Iris has Student Help hours Thursday 10a-12p
  - Shikha’s Student Help Hours are canceled unless otherwise noted

Please fill out the CS134 Remote Questionnaire (click here)
The CS department has a page of Resources for Remote Work.

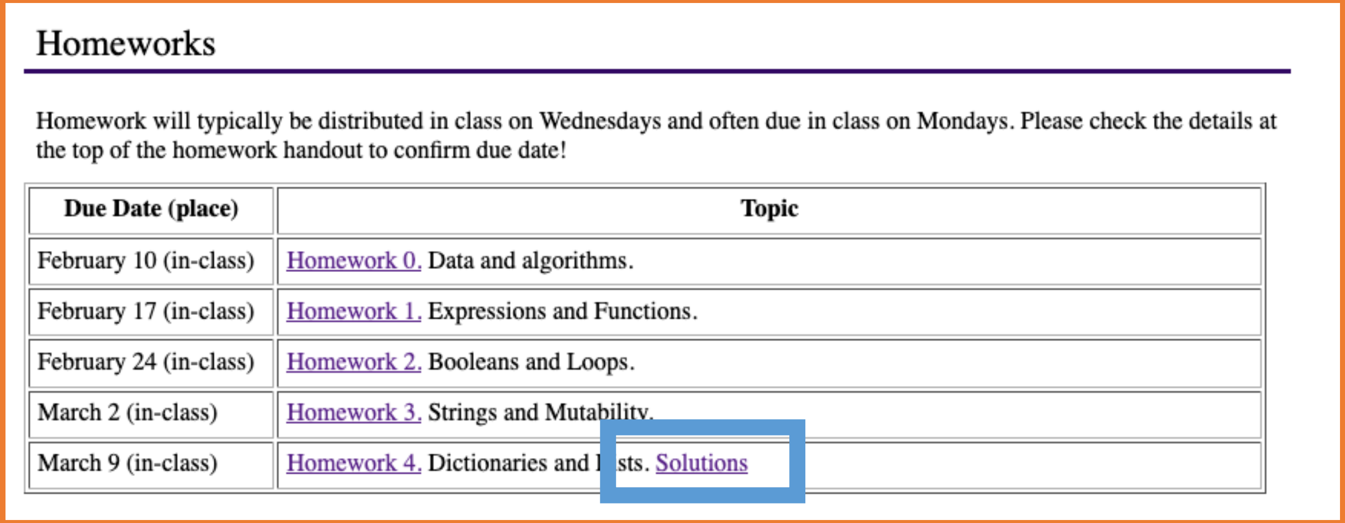
Please bring your personal laptop to class on Friday
so we can try to get you set-up.
You might be able to borrow a laptop longterm from the library.

Midterm Exam is Thursday, March 12

- TPL 203
- 5:45pm-7:45pm OR 8-10pm

- Closed book exam
- Review your homeworks! Slides! Labs!
- POGILs/Jupyter Notebooks!

- HW4 Solutions posted

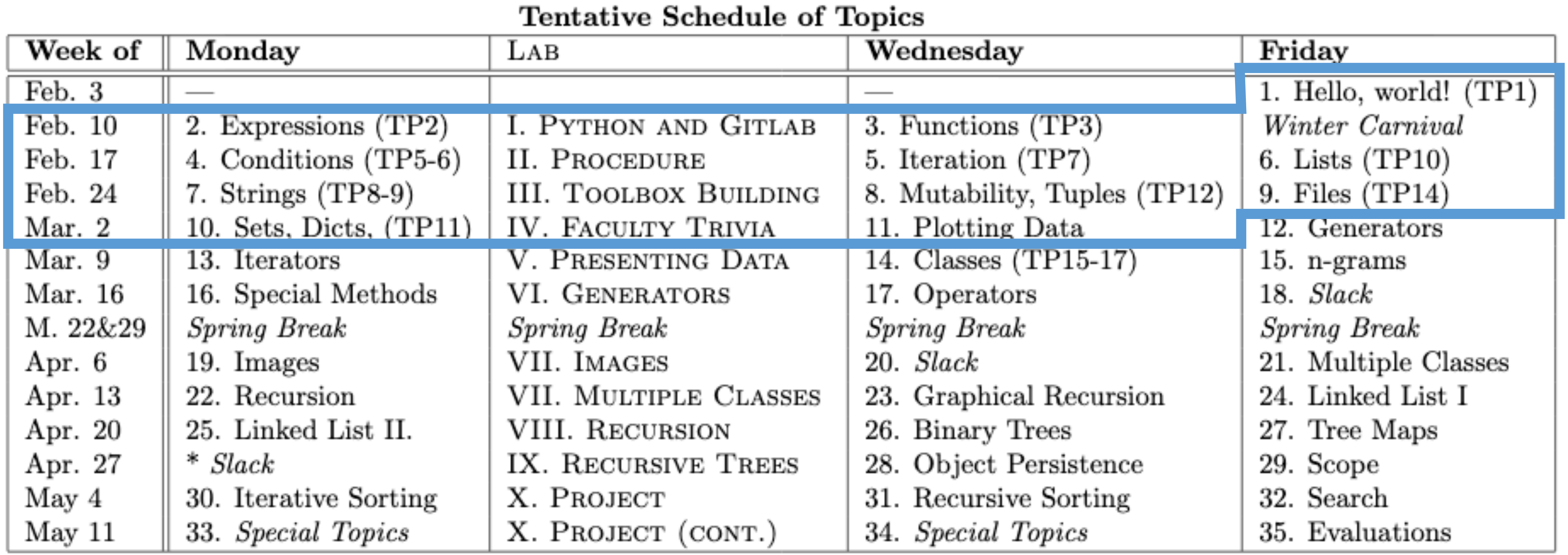


Homeworks

Homework will typically be distributed in class on Wednesdays and often due in class on Mondays. Please check the details at the top of the homework handout to confirm due date!

Due Date (place)	Topic
February 10 (in-class)	Homework 0. Data and algorithms.
February 17 (in-class)	Homework 1. Expressions and Functions.
February 24 (in-class)	Homework 2. Booleans and Loops.
March 2 (in-class)	Homework 3. Strings and Mutability.
March 9 (in-class)	Homework 4. Dictionaries and Lists. Solutions

MIDTERM EXAM IS THURSDAY, MARCH 12



Tentative Schedule of Topics

Week of	Monday	Lab	Wednesday	Friday
Feb. 3	—		—	1. Hello, world! (TP1)
Feb. 10	2. Expressions (TP2)	I. Python and Gitlab	3. Functions (TP3)	*Winter Carnival*
Feb. 17	4. Conditions (TP5-6)	II. Procedure	5. Iteration (TP7)	6. Lists (TP10)
Feb. 24	7. Strings (TP8-9)	III. Toolbox Building	8. Mutability, Tuples (TP12)	9. Files (TP14)
Mar. 2	10. Sets, Dicts, (TP11)	IV. Faculty Trivia	11. Plotting Data	12. Generators
Mar. 9	13. Iterators	V. Presenting Data	14. Classes (TP15-17)	15. n-grams
Mar. 16	16. Special Methods	VI. Generators	17. Operators	18. *Slack*
M. 22&29	*Spring Break*	*Spring Break*	*Spring Break*	*Spring Break*
Apr. 6	19. Images	VII. Images	20. *Slack*	21. Multiple Classes
Apr. 13	22. Recursion	VII. Multiple Classes	23. Graphical Recursion	24. Linked List I
Apr. 20	25. Linked List II.	VIII. Recursion	26. Binary Trees	27. Tree Maps
Apr. 27	* *Slack*	IX. Recursive Trees	28. Object Persistence	29. Scope
May 4	30. Iterative Sorting	X. Project	31. Recursive Sorting	32. Search
May 11	33. *Special Topics*	X. Project (cont.)	34. *Special Topics*	35. Evaluations

TOPICS

- Expressions. Booleans. If statements/conditionals. Simplification. Int()
- Strings. Split(), iterating over, slicing, concatenation, isupper(), lower(), string methods, str()
- Functions. Writing your own, Value returning & None-returning, Helper functions
- Lists. Slicing, indexing, iterating over, nested lists, sort() and sorted(), list comprehensions, len()
- Dictionaries. .get() method, keys, values
- Loops. Nested, for, while, in

- Debugging
- Set()
- File Reading

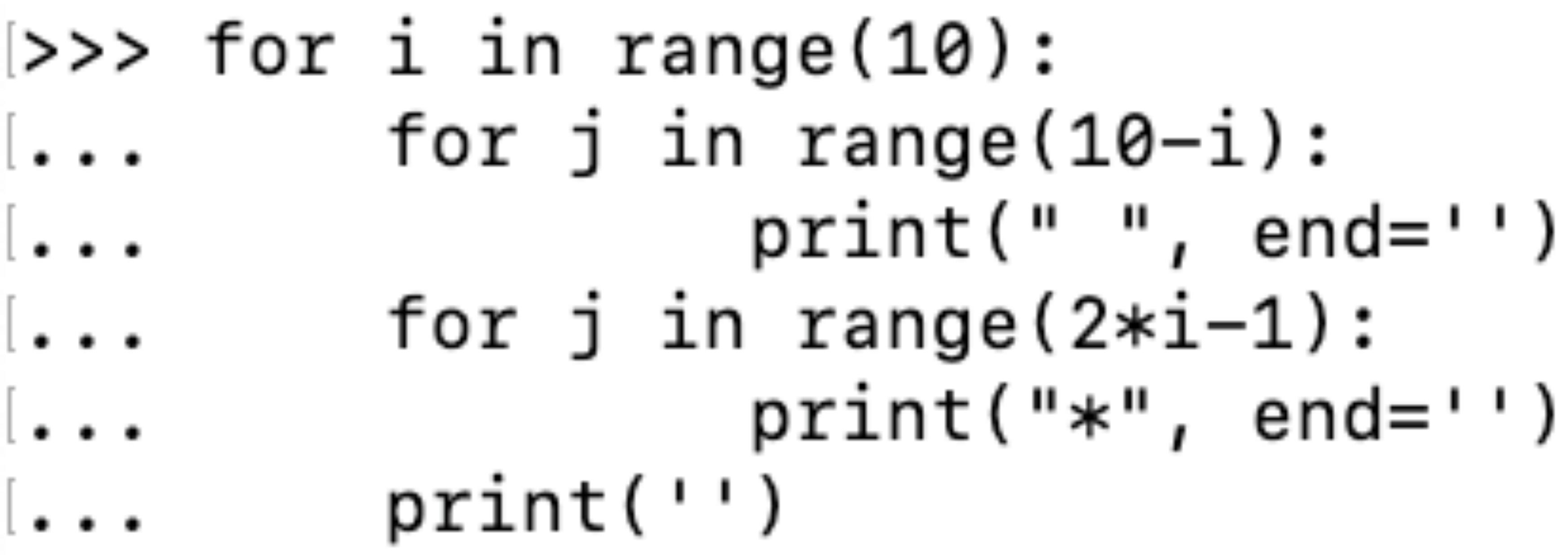
- Lambda
- Doctests
- __all__

Topics

- *Homework 1*: Expressions & Functions, return & print
- *Homework 2*: booleans & loops over sequences, simplifying conditionals, list indexing
- *Homework 3*: strings & mutability
- *Homework 4*: Tuples, Dict (get), list comprehension, lambda sorting
- *From labs:*
  - Writing functions, File reading; Strip, split; Sorting, strings; Len; Finding max; Counters in loops; Doctests, __all__, modules/scripts, if __name__=='__main__'
- Pretty much everything up to and including Lab 4 & Homework 4

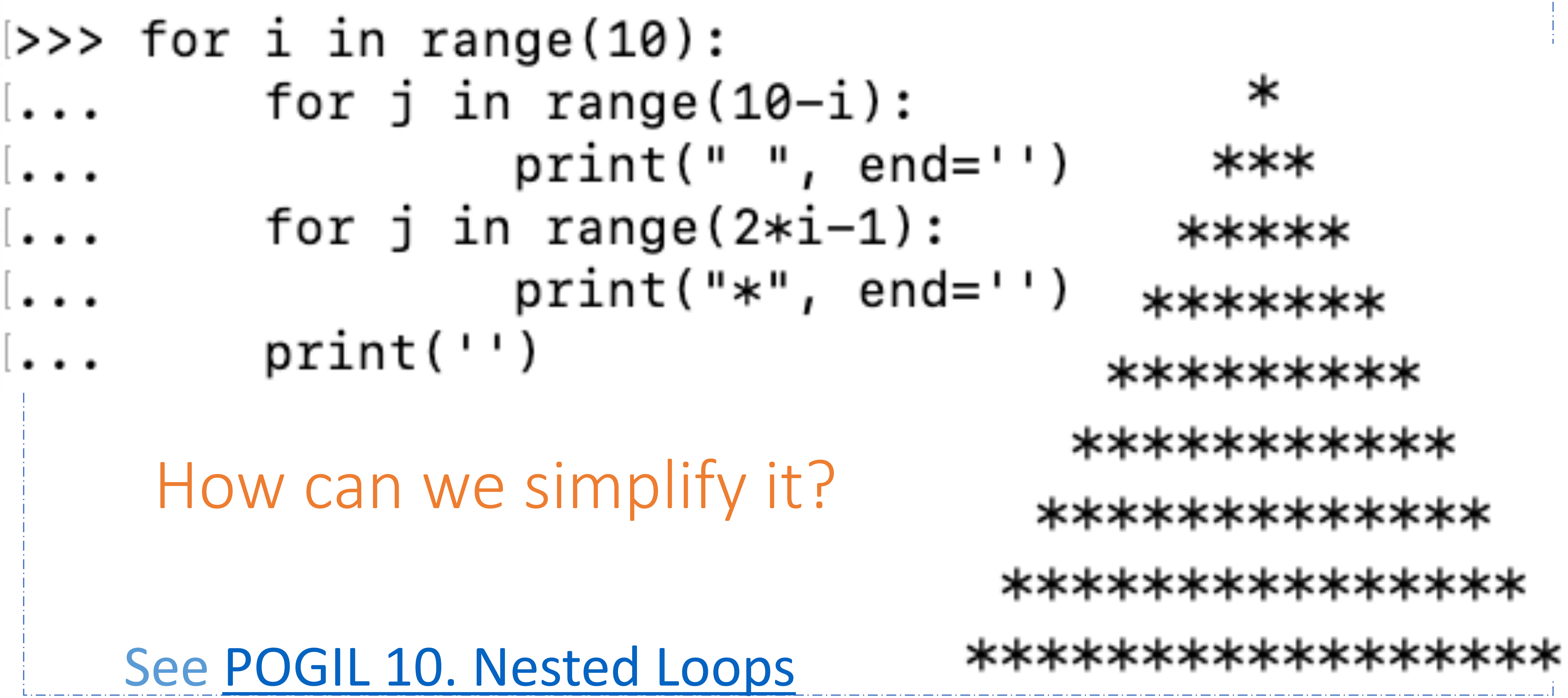
QUESTIONS?

What does this do?



```
>>> for i in range(10):
...     for j in range(10-i):
...             print(" ", end='')
...     for j in range(2*i-1):
...             print("*", end='')
...     print('')
```

What does this do?



```
>>> for i in range(10):
...     for j in range(10-i):
...             print(" ", end='')
...     for j in range(2*i-1):
...             print("*", end='')
...     print('')
```

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

How can we simplify it?

See POGIL 10. Nested Loops

st.islower()

```
>>> s = 'heLLo GooDbYe'
>>> for c in s:
...     if c.islower():
...             print(c)
...
```

```
h
e
o
o
o
b
e
```

List Comprehensions

- def sized(num, wdList):
    - ””” Returns a list of words from wdList length num “””
    - return [wd for wd in wdList if len(wd) == num]

```
>>> sized(6, ['123456', 'hey', 'ho', 'letsgo’])
['123456', 'letsgo’]
```

- def appendArghEachWord(wdList):
    - return [wd + ‘argh’ for wd in wdList]

```
>>> appendArghEachWord(['yo','ho','a','pirates','life'])
['yoargh', 'hoargh', 'aargh', 'piratesargh', 'lifeargh']
```

Lambda

```
>>> lst = [ [1,9], [2,8], [3,7] ]
>>> def bySecond(pr):
...     return pr[1]

# Sorts by the 1th item:
>>> sorted(lst, key=bySecond)
[[3, 7], [2, 8], [1, 9]]

# Also sorts by the 1th item:
>>> sorted(lst, key=lambda pr: pr[1])
[[3, 7], [2, 8], [1, 9]]
```

sorted’s key parameter lets us specify how to sort a sequence.

Lambda functions are anonymous, single use functions

A good choice for specifying how to sort items in a sequence

Dictionary .get()

```
>>> d = dict()
>>> type(d)
<class 'dict'>
>>> d['month'] = 'march'
>>> d['day'] = 9
>>> d
{'month': 'march', 'day': 9}
>>> d.get('month')
'march'
```

```
>>> d.get('day')
9
>>> d.get(‘NOPE')
>>> d.get(‘NOPE', 'ERROR')
'ERROR'
>>> d
{'month': 'march', 'day':
9}
```

.get() provides a default value to return when that key doesn’t exist in the dictionary

Using .get() to store counts

```
>>> d = {'apple’: 1}
>>> d['grape’] = 2}
>>> d.get('apple', 0)+1
2
>>> d
{'apple': 1, 'grape': 2}
>>> d['apple'] = d.get('apple', 0)+1
```

```
>>> d
{'apple': 2, 'grape': 2}
>>> d['tomato'] =
d.get('tomato', 0)+1
>>> d
{'apple': 2, 'grape': 2,
'tomato': 1}
```

.get() is very useful when you want to update the value of a key that may not exist (when updating lotsa keys)

Tuples as Dictionary Keys

```
>>> d = dict()
>>> d['apple'] = 1
>>> d['grape'] = 2
>>> d[(0,1)] = ' whatev'
>>> d[(0,1)]
' whatev '
```

```
>>> d[0]
Traceback (most recent call
last): KeyError: 0
>>> d[0] = 'hello!'
>>> d
{'apple': 1, 'grape': 2, (0,
1): ' whatev ', 0: 'hello!'}
>>> 0 == (0,1)
False
```

The dict key is the entire tuple! One element of tuple is a different key!

None & Value Returning Functions

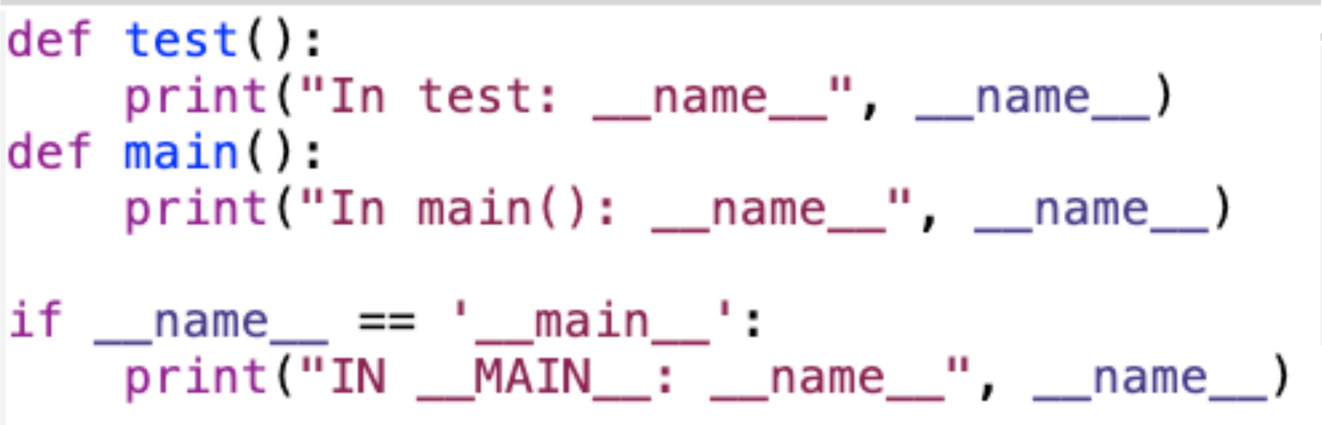
```
>>> a = lst.append(6)
>>> lst
[0, 2, 3, 4, 6, 6]
>>> a
>>> print(a)
None
```

Storing what’s returned by append make the list ‘a’ None

```
>>> lst = [0,2,3,4]
>>> lst.append(6)
>>> lst
[0, 2, 3, 4, 6]
>>> type(lst)
<class 'list'>
```

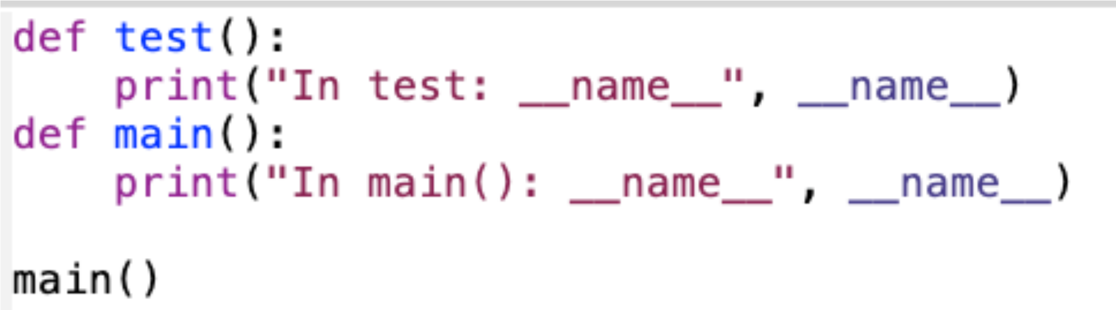
Appending to lst does not make the lst None!

If _name_=‘_main_’ vs main() - script



```python
def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

if __name__ == '__main__':
    print("IN __MAIN__: __name__", __name__)
```

```
-> python3 test.py
IN __MAIN__: __name__ __main__
```

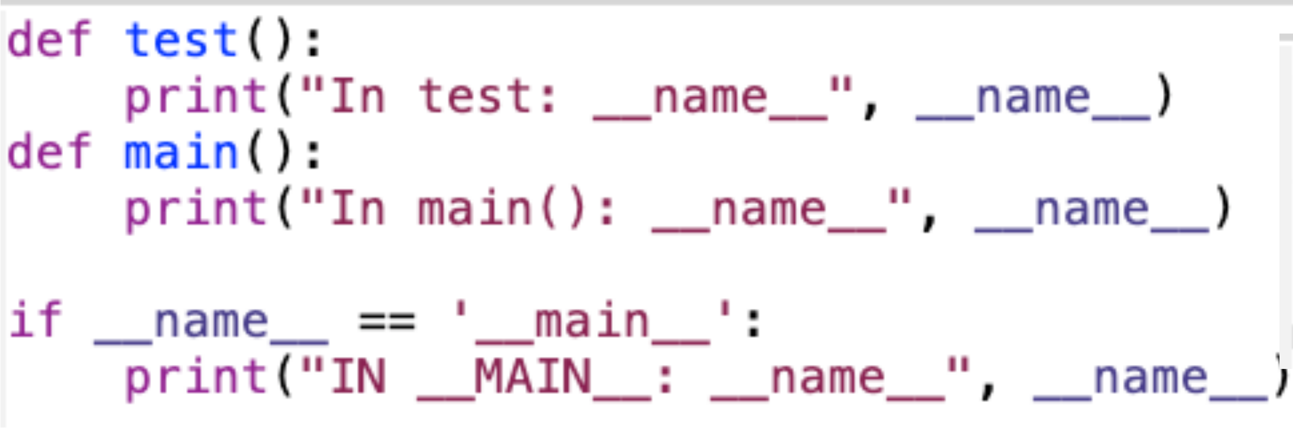


```python
def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

main()
```

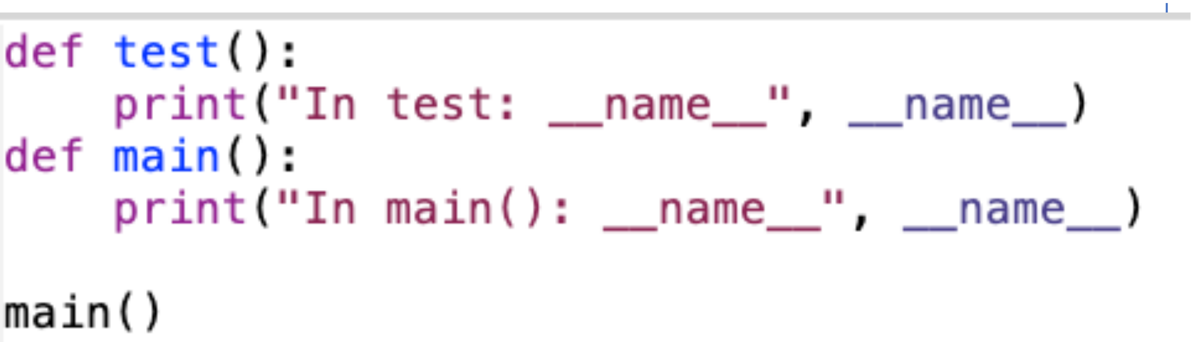
```
-> python3 test.py
In main(): __name__ __main__
```

When run as script, __name__ == ‘__main__’

If _name_='_main_' vs main() - import



```
def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

if __name__ == '__main__':
    print("IN __MAIN__: __name__", __name__)
```



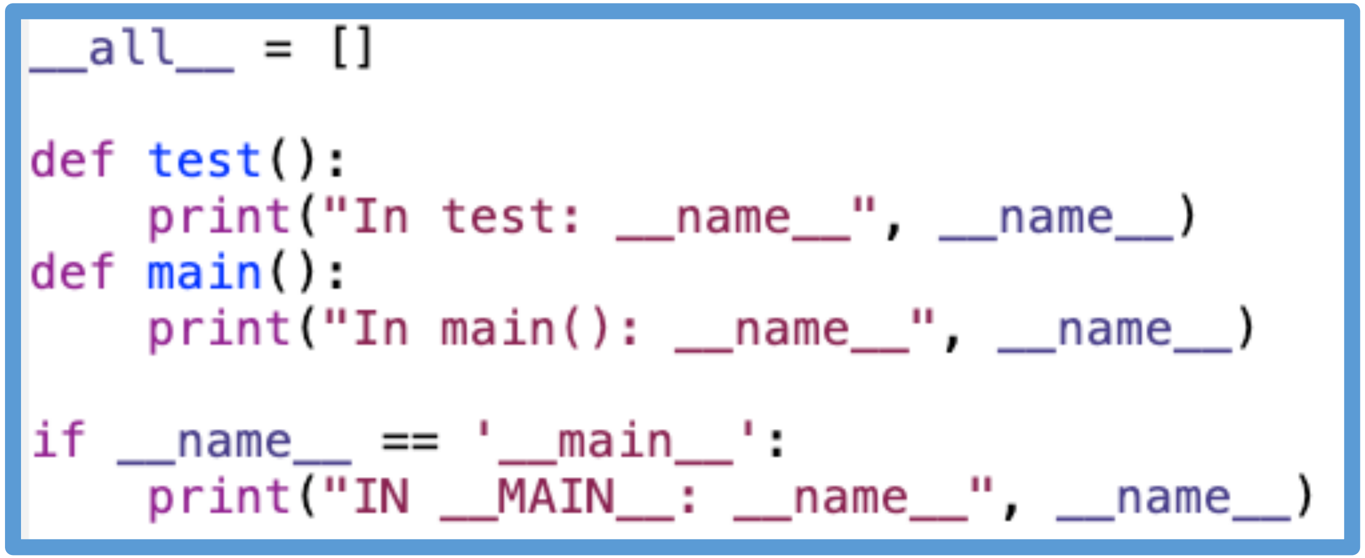
```
def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

main()
```

```
-> python3
>>> import test
>>> test.main()
In main(): __name__ test
```

```
-> python3
>>> import test
In main(): __name__ test
```

On right: main() is called even when you import

__all__



```python
__all__ = []

def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

if __name__ == '__main__':
    print("IN __MAIN__: __name__", __name__)
```

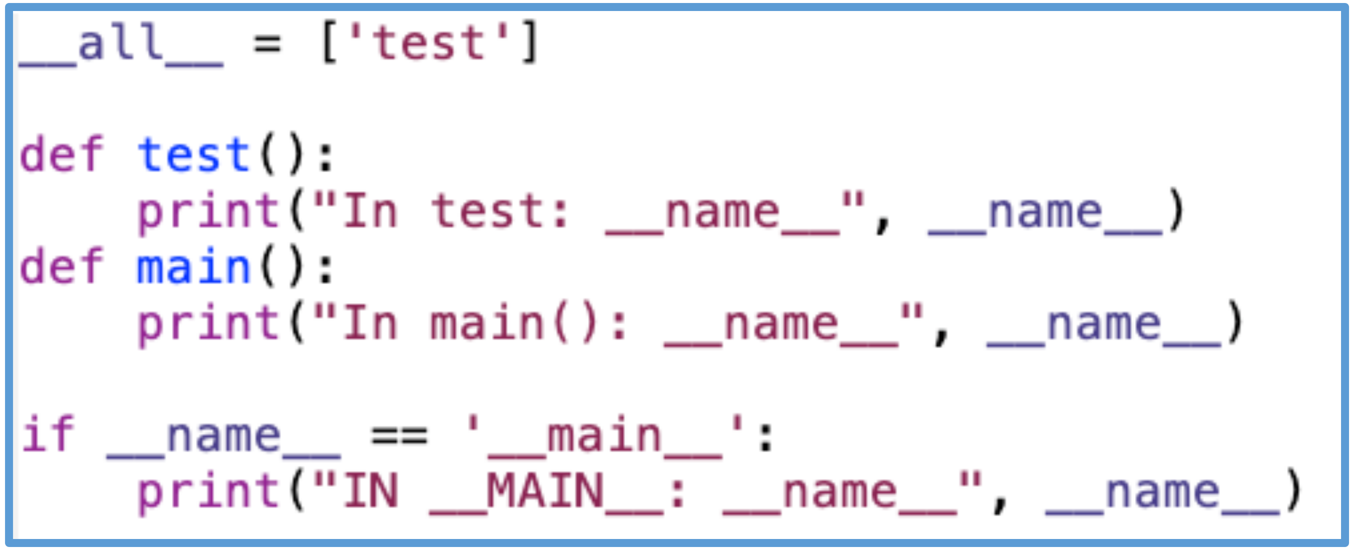
```
>>> from test import *
>>> test()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'test' is not defined
```

When __all__ is empty, you have to:

`from test import test`

SEPARATELY!!

__all__



```
__all__ = ['test']

def test():
    print("In test: __name__", __name__)
def main():
    print("In main(): __name__", __name__)

if __name__ == '__main__':
    print("IN __MAIN__: __name__", __name__)
```

Put public functions in __all__ and you can use them without separate import statements

```
>>> from test import *
>>> test()
In test: __name__ test
```

Question About Aliasing

- Looked at Iris' slides on mutability from February 26
- https://williams-cs.github.io/cs134-s20-www/iris-lectures/Lecture08-tuples.pdf

HW3 Question 2d

```
# without aliasing
>>> nl = [[1,2],[3,4]]
>>> nl.append([3,4])
>>> nl[2][1]=6
>>> nl
[[1, 2], [3, 4], [3, 6]]
```

```
# with aliasing
>>> nl = [[1,2],[3,4]]
>>> nl.append(nl[1])
>>> nl[2][1]=6
>>> nl
[[1, 2], [3, 6], [3, 6]]
```

Note the final values! Why is the 2nd one different?

Aliasing points them to same balloon!

Playing w Sets

Getting the unique values from sequences

```
>>> l = [2,3,4,5,5,5,5,5,5,6,2]
>>> set(l)
{2, 3, 4, 5, 6}
>>> set(list(range(5)))
{0, 1, 2, 3, 4}
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> t = (5,5,5,3,2,2)
>>> set(t)
{2, 3, 5}
>>> set('aaaabbc')
{'b', 'a', 'c'}
```

Sets & Mutability

```
>>> a = {1,2,3}
>>> b = a
>>> b == a
True
>>> b is a
True
>>> a.add(4)
>>> a
{1, 2, 3, 4}
>>> b
{1, 2, 3, 4}
```

```
>>> l = [1,2,3]
>>> m = [1,2,3]
>>> l.append(4)
>>> l == m
False
>>> l is m
False
>>> m
[1, 2, 3]
>>> l
[1, 2, 3, 4]
```

Reading Files

Opens the file

Filename as a string

Opens filename calls it fin

- `with open(‘prideprejudi.txt’) as fin:`
  - `for line in fin:` For each line in our file, fin
    - `pass` Does nothing, why’s it here? What should be here?

Once we leave the “with” indentation, the file is closed!

- `# file is implicitly closed`

FILES MUST BE OPENED, READ, AND THEN CLOSED

Writing Files

Specifies mode. w means?

What if we had ‘r’ here?

Opens the file

Filename as a string

Opens filename calls it fout

- with open(‘newFile.txt’, ‘w’) as fout:
    - fout.write(“Hello!!”)

Writes to the file, ”Hello!!”

    - for item in mylist:
        - fout.write(item)

Writes an entire list to a file

Once we leave the “with” indentation, the file is closed!

- # file is implicitly closed

If unable to use the ‘with’ keyword, can also use fout.close() to explicitly close file

FILES MUST BE OPENED, WRITTEN, AND THEN CLOSED

Lab 05: Matplotlib

- Matplotlib functions will not be on the exam

- But knowing how to manipulate dictionaries, sort lists, move data around, write your own doctests, read from a CSV are very *important*

Leftover Slides