# UCLA CS35L

Week 4

Monday

## Reminders

- Assignment 3 homework is due this Friday (4/24)
- Assignment 4 homework is due next Friday (5/1)

- Anonymous feedback for Daniel - https://forms.gle/tZwuMbALe825DBVn8

# Python

## Python History

- Developed Guido van Rossum in the Netherlands
  - Guido was named BDFL or Benevolent Dictator For Life of Python
  - Has since helped shape and lead Python’s development

- Python was named due to Guido being a fan of Monty Python’s Flying Circus (A British Comedy Show)

- There is a “Zen of Python” to describe the language philosophy
  - Big focus on readability

## Why is Python So Popular?

- Python is accessible. Code is very readable and easy to write
    - Language features like dynamic typing and indentation help
- Many powerful libraries and tools
    - Data science, Web Dev, etc.
- Good developer support and active community involvement in open-source development

## Note on Python Versions

- Python3 is the current version (specifically 3.8) and nearly all new Python development is done with Python3
- Python2 was popular as well, but became End-of-Life at the start of 2020.
    - There is still plenty of now “legacy” applications still running Python2
- We will use Python3, and that does add additional features

## Python Coding Basics

- Python in an interpreted language – no compilers
- Python is dynamically typed
  - You do not have to declare the type of your variable, and there is no compiler to check for errors related to that (type checking)
  - Offer goods flexibility, but can lead to potential bugs and unsafe code in the long-term
- Typical uses
  - Prototyping and Scripting
  - Research – ML, Data Science
  - Leetcode
  - Some web-related code (Django/Flask)

## Python Under the Hood

- The reference implementation of Python is done in C (CPython)
    - Note the difference between the implementation of a language and the actual language itself
- Python behind the scenes will take care of details like memory management.
- There is a Global Interpreter Lock (GIL) which then limits your Python code to one thread at a time, to simplify low-level implementation details.
    - Limits Python performance in parallelized applications and multi-core systems
    - We can use separate processes, instead of separate threads to get parallel performance. But that is more complicated since processes don’t share memory space

## Data Types – Mutability and Immutability

- Some immutable types
  - Int
  - Float
  - Bool
  - Tuple
  - Str

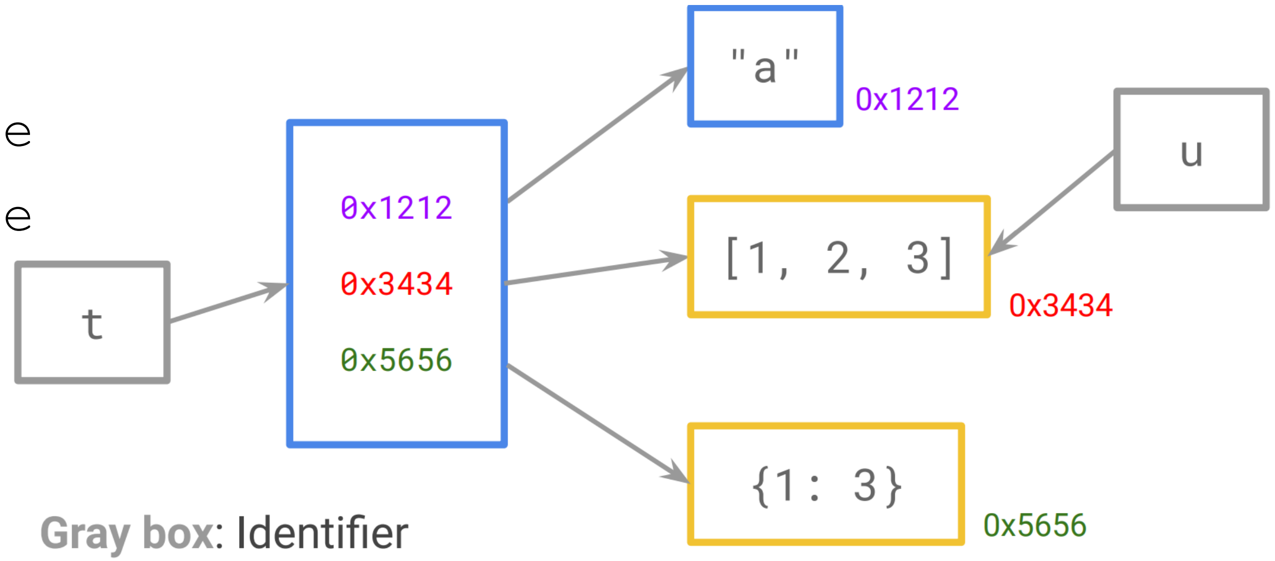
- Some mutable types
  - List
  - Dict
  - Set

NOTE – Python is dynamically typed. So you do not need to declare these types when you create a variable

## Python Examples

# Mutability and Immutability

```
u = [1, 2, 3]
t = ('a', u, {1: 3})
t[0] = 'bc'          t is immutable
t[1] = [3,4]         t is immutable
t[1].append('c')     t[1] is mutable
t[2][2] = 3          t[2] is mutable
```



Gray box: Identifier
Blue: Immutable data
Gold: Mutable data

## Printing

- Print() function
  - Print(‘hello_world’)
- Multiple ways to print variables from print function.
  - My preferred way is to use the f-string (only in Python 3.6+)
  - Print(f “{var1} {var2}”)

- Other ways include:
  - String concatenation (+ sign)
  - .format()
  - % identifiers like in C

## Conditionals

- Have access to typical operators from C
  - `<, <=, >, >=, ==, !=`
- And/Or/Not are just spelled out
  - `if cond1 and cond2`
  - `if cond3 or cond4`
  - `if not (cond5)`
- Can run simplified checks to see if values are in list/set
  - `if val in list`
  - `If substring in bigger_string`

## is vs ==

- is compares memory addresses
- == invokes the obj._eq_ method
  - Primarily used for value comparisons
- “is None” is preferred over “== None” if the goal is to test for None

## Lists

- Python lists act like C++ Vectors
  - They track their own size
  - You can add/remove elements, and the list will resize as necessary

- You can access elements like in C++
  - `my_list[index]`

## Slicing

```
l = list(range(6))  # i.e. l = [0, 1, 2, 3, 4, 5]
l[-2]               # second to last, i.e. 4
l[:3]               # the first 3 items
l[2:]               # starting with index 2, i.e. [2, 3, 4, 5]
l[3:4]              # 4 is exclusive, i.e. [3]
l[1:4:2]            # the third one is stride, i.e. [1, 3]
l[::2]              # omission means default start and end, i.e. [0, 2, 4]
l[:-2]              # skipping the last two, i.e. [0, 1, 2, 3]
The same works with strings.
s = "012345"
s[2:4] # gives '23'
s[-1] # gives '5'
Note that there is no char type in Python, '5' is a string of length 1.
```

## Helpful Python String Methods

https://docs.python.org/3.7/library/stdtypes.html#string-methods

`string.split()`
If no argument is provided, will split the string by whitespaces (space + tabs)
`'1,2,3'.split(',')` splits by comma

`str.strip([chars])`
Returns a copy of the string with the leading and trailing characters in `[chars]` removed.

```
       ‘     spacious    ’.strip() -> ‘spacious’
       ‘www.example.com’.strip(‘cmowz.’) -> ‘example’
```

## Range and For-in

- Range is a function to generate an immutable sequence type, useful for loops.

```
for i in range(5)
    print(i)
#prints 0, 1, 2, 3, 4
```

- Can also access contents of a list/set/dict directly

```
my_list = [10, 15, 20, 25]
for num in my_list
    print(num)
#prints 10, 15, 20, 25
```

## Functions

```python
def hello_word():
    print('hello_word')
```

```python
def get_second_arg_plus_one(arg1, arg2):
    print(arg1)
    return arg2 + 1
```

# Indentation and Whitespace

- Python is very sensitive to indentation and whitespace
- Each code block needs to be indented to be grouped together
  - The python interpreter will assign scope that way
  - Try to only use tabs or spaces, a mixture of both can mess up the grouping
- Statements the begin a new block (if, loops, functions) need a :
- Note – no semicolons required!

```python
def get_second_arg_plus_one(arg1, arg2):
    print(arg1)
    return arg2 + 1
```

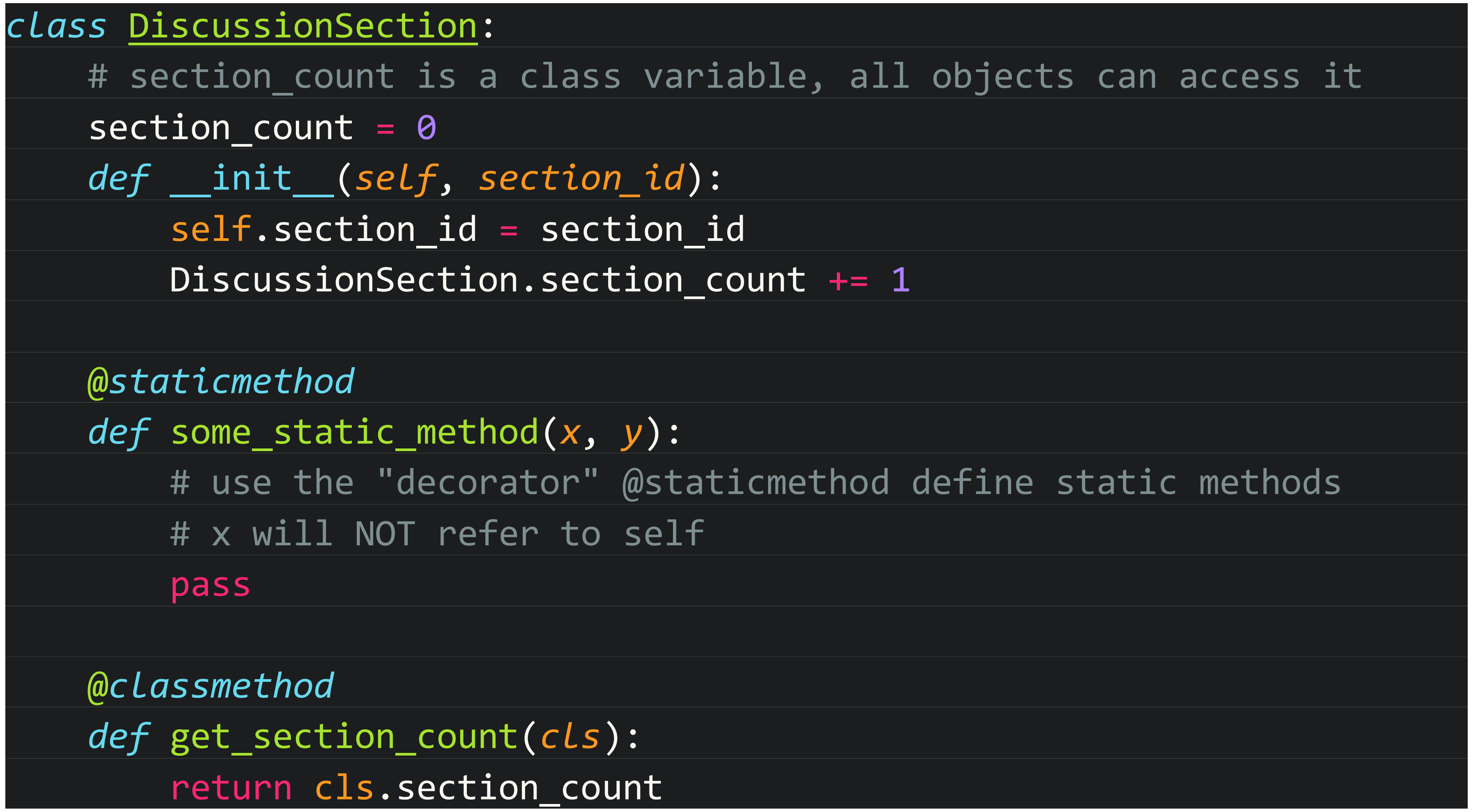
## Naming Convention

```
"""
UpperCamelCase for class names
CAPITALIZED_WITH_UNDERSCORES for script level constants
snake_case for everything else
"""

CONST_VAR = 3

def functions_snake_case():
    # this is how we define a function in Python
    variables_snake_case = 1
class ClassName:
    # this is how we define a class in Python
    def methods_snake_case(self):
        print("descriptive names")
```

# Classes

```
class DiscussionSection:
    def __init__(self, section_id, ta):
        self.section_id = section_id
        self.ta = ta
        self.students = {}
    def add_student(self, name, uid):
        self.students[name] = uid


def run():
    section = DiscussionSection(2, "Aaron")
    section.add_student("some_student", 123456789)
    section.add_student("some_student", 123456789)
```

## Classmethod and Staticmethod

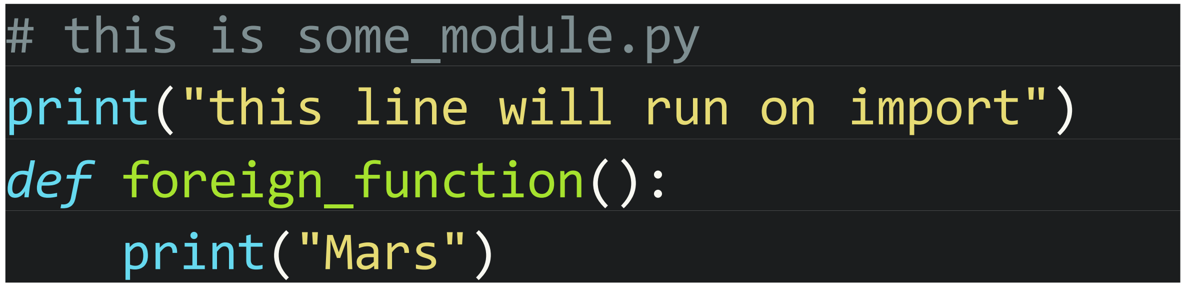


```python
class DiscussionSection:
    # section_count is a class variable, all objects can access it
    section_count = 0
    def __init__(self, section_id):
        self.section_id = section_id
        DiscussionSection.section_count += 1

    @staticmethod
    def some_static_method(x, y):
        # use the "decorator" @staticmethod define static methods
        # x will NOT refer to self
        pass

    @classmethod
    def get_section_count(cls):
        return cls.section_count
```

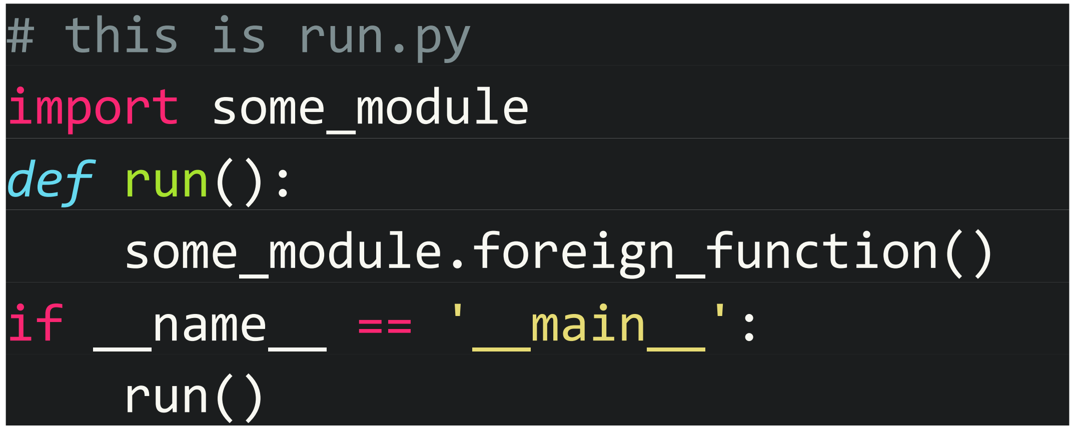
## Notes on Classes

- ‘self’
  - The first argument of any class function will always be the object the function is called on.
  - This variable can be called anything, but convention is to use ‘self’
  - To change an object’s variable x. You must use self.x
- Staticmethod
  - Used if you want to write a method that does not rely on class variables
  - The first argument than will NOT refer to self
- Classmethod
  - Similar to a static method, but able to access class variables
  - Use cls argument

## Import



```python
# this is some_module.py
print("this line will run on import")
def foreign_function():
    print("Mars")
```



```python
# this is run.py
import some_module
def run():
    some_module.foreign_function()
if __name__ == '__main__':
    run()
```

- When some_module is imported, the interpreter pauses run.py and reads all the lines in some_moduly.py
  - def's are definitions to be remembered
  - Other lines are executed

- Another variant of import is:
  - from some_module import foreign_function

# Python Memory Model

Pass by Object Reference

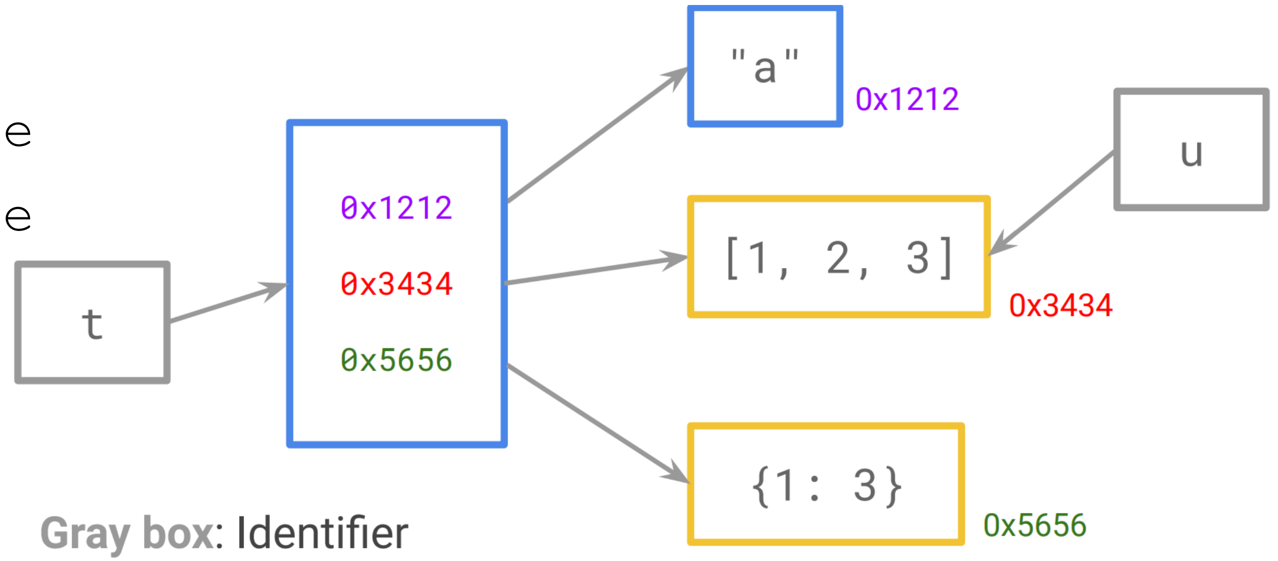
# Python Memory Management

- Everything is Pass-by-object reference
  - Distinct from Pass-by-value and Pass-by-reference that C++ uses
- When we pass a variable in Python, we pass an object that holds the same reference to the original variable.
  - If we modify the same variable than that modification appears in the original
  - But we can reassign to a new object without affecting the original

- Good website to go over all the differences
  - https://robertheaton.com/2014/02/09/pythons-pass-by-object-reference-as-explained-by-philip-k-dick/

# Mutability and Immutability

```
u = [1, 2, 3]
t = ('a', u, {1: 3})
t[0] = 'bc'          t is immutable
t[1] = [3,4]         t is immutable
t[1].append('c')     t[1] is mutable
t[2][2] = 3          t[2] is mutable
```

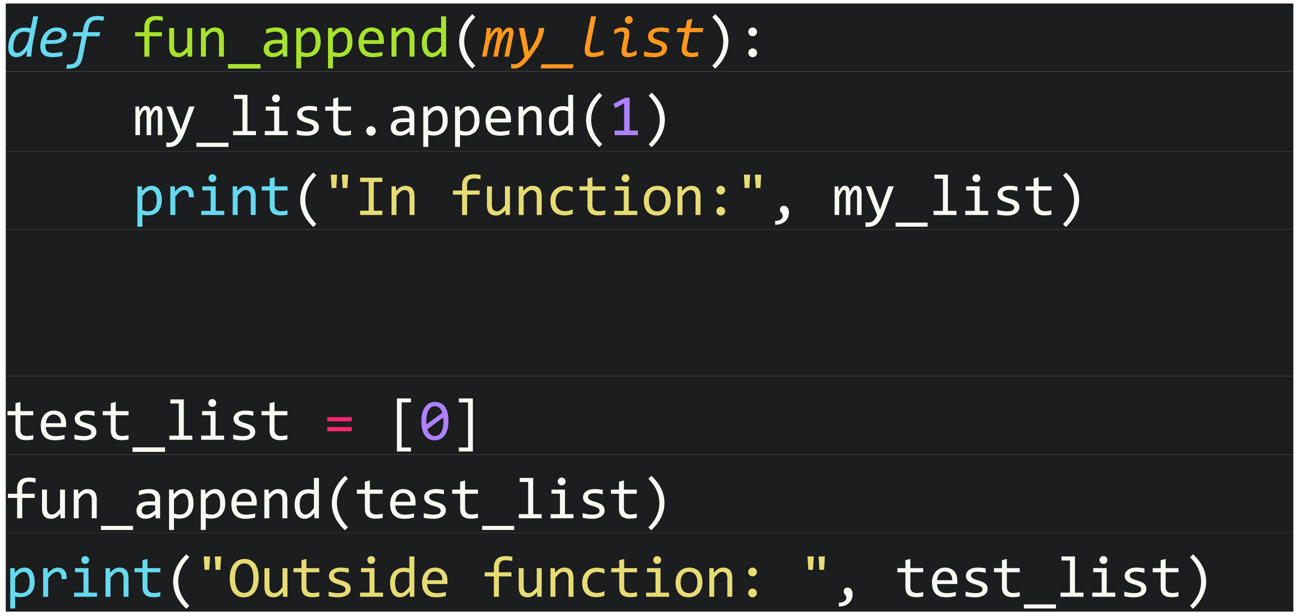


**Gray box**: Identifier
**Blue**: Immutable data
**Gold**: Mutable data

# Memory Management Example 1

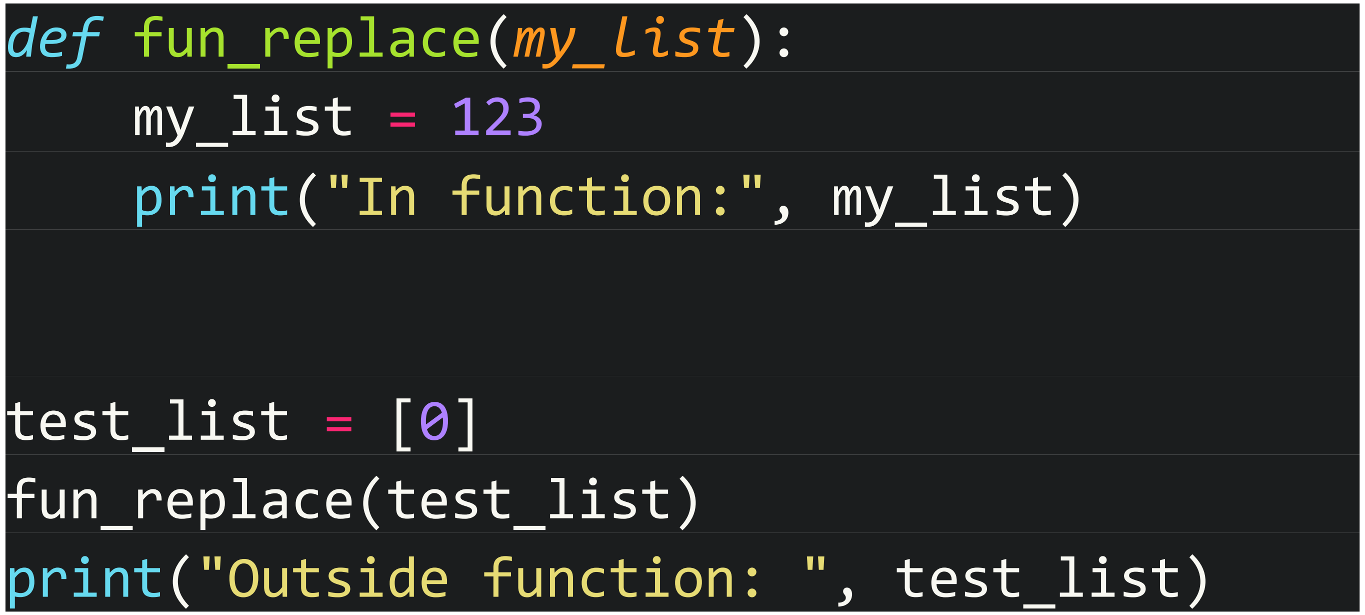
```
some_guy = 'Fred'

first_names = []
first_names.append(some_guy)
#first_names = [Fred]

another_list_of_names = first_names
#another_list_of_names = [Fred]
another_list_of_names.append('George')
#another_list_of_names = [Fred, George]
some_guy = 'Bill'
#some_guy = Bill

print (some_guy, first_names, another_list_of_names)
#Bill ['Fred', 'George'] ['Fred', 'George']
```

## Memory Management Example 2



```python
def fun_append(my_list):
    my_list.append(1)
    print("In function:", my_list)


test_list = [0]
fun_append(test_list)
print("Outside function: ", test_list)
```

## Memory Management Example 3



```python
def fun_replace(my_list):
    my_list = 123
    print("In function:", my_list)


test_list = [0]
fun_replace(test_list)
print("Outside function: ", test_list)
```

- Only difference is from previous example is `my_list = 123`

## Shallow vs Deep Copy

- A **shallow copy** constructs a new compound object and then (to the extent possible) inserts references into it to the original objects
- A **deep copy** constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original

```python
from copy import copy, deepcopy

a = [1, 2, [3, 4]]  # original list
b = copy(a)         # shallow copy (reference to sublist [3, 4])
c = deepcopy(a)     # deep copy (creates copy of sublist [3, 4])
```

## Copy Example

```python
from copy import copy
from copy import deepcopy

a = [1,2,[3,4]]
b = copy(a)
c = deepcopy(a)

b[0] = 0
b[2][0] = 10

print(a)
print(b)
print(c)
```

- Create list a = [1, 2, [3,4]]
- b is a shallow copy of a, b = [1, 2, [3,4]]
- c is a deep copy of a, c = [1, 2, [3,4]]

- b[0] is assigned 0. Due to pass by object reference, this reassignment only changes b
- b[2][0] is assigned 10. The reference to the list object [3,4] is shared, so both a and b are changed.

## Python (Past this Course)

## Python Environment Setup

- Most Linux distributions come with Python already available. But if you start developing more in Python, you’ll probably want to look into tool and managers for Python environments
  - Conda
  - Pip and Venv
  - I think Conda is becoming the most popular, but both are ways to manage Python packages and Virtual Development environments

- A common way to access Conda is by installing ‘Anaconda’.
  - Anaconda is a distribution that includes Conda and other Python libraries

## Python for Data Science/ML

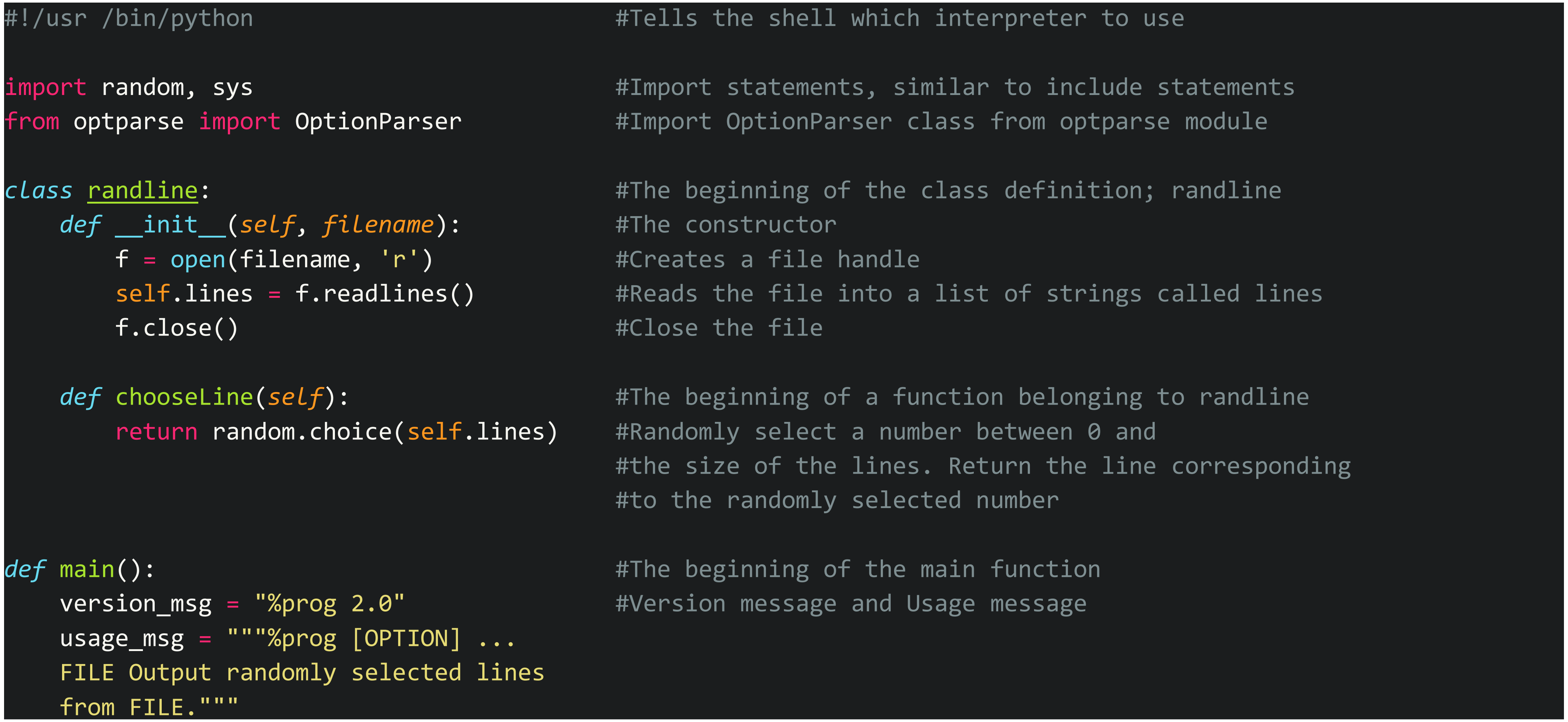
- Python is the premier language of choice for Data Science and ML
- Some tools or libraries that you should look into if interested in that world:
  - Numpy
  - Scipy
  - Pandas
  - Keras
  - Jupyter Notebooks
- These are all accessible via Anaconda

## Python for Web Dev

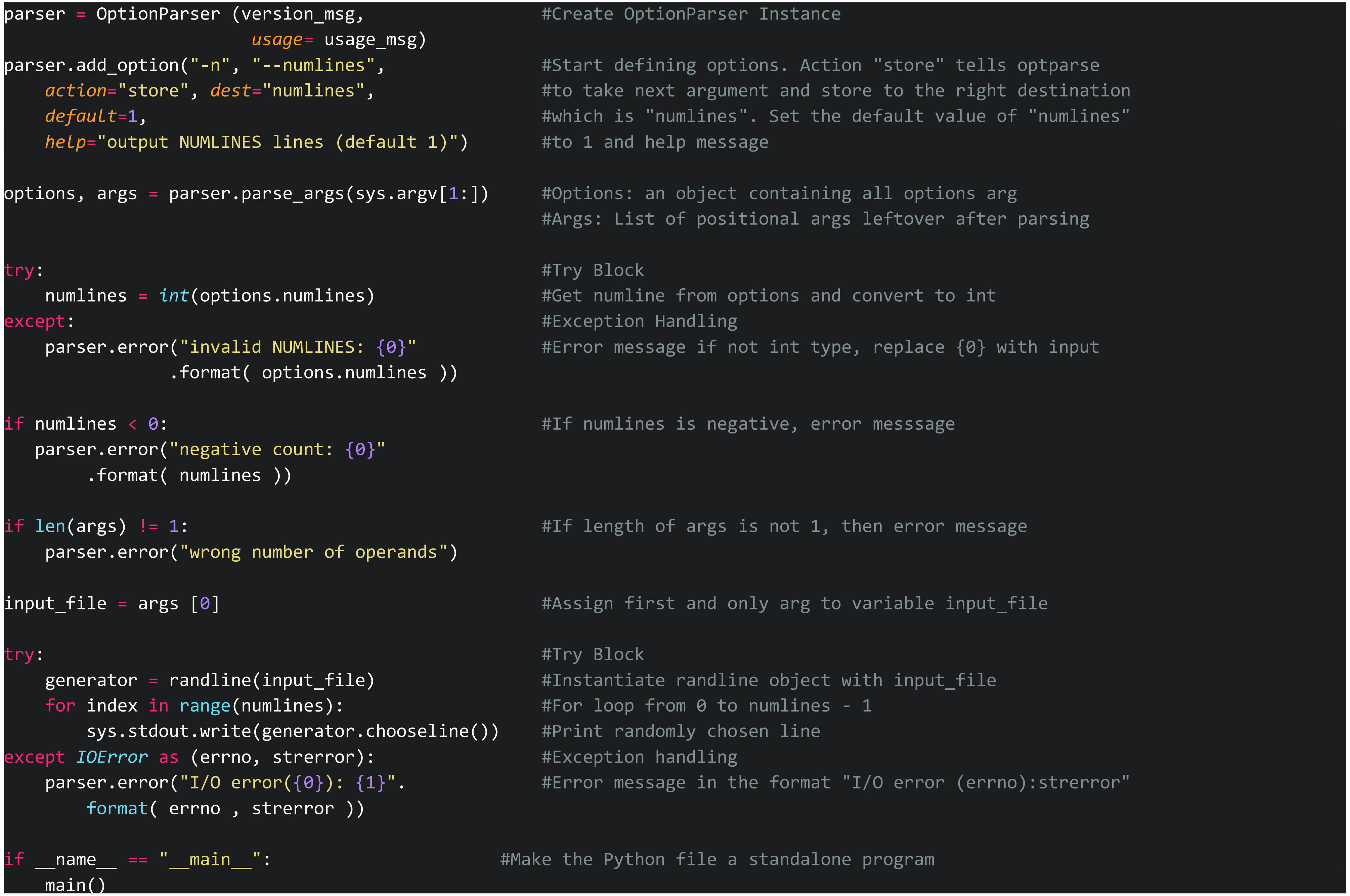
- Python is single-threaded, but can support asynchronous calls. So it is commonly used in some website backends.
- Most popular frameworks
  - Django
  - Flask
- Both of those are included in Anaconda

## Python Homework

## Python Lab Walkthrough



```
#!/usr /bin/python                         #Tells the shell which interpreter to use

import random, sys                         #Import statements, similar to include statements
from optparse import OptionParser          #Import OptionParser class from optparse module

class randline:                            #The beginning of the class definition; randline
    def __init__(self, filename):          #The constructor
        f = open(filename, 'r')            #Creates a file handle
        self.lines = f.readlines()         #Reads the file into a list of strings called lines
        f.close()                          #Close the file

    def chooseLine(self):                  #The beginning of a function belonging to randline
        return random.choice(self.lines)   #Randomly select a number between 0 and
                                           #the size of the lines. Return the line corresponding
                                           #to the randomly selected number


def main():                                #The beginning of the main function
    version_msg = "%prog 2.0"              #Version message and Usage message
    usage_msg = """%prog [OPTION] ...
    FILE Output randomly selected lines
    from FILE."""
```



```
parser = OptionParser (version_msg,                 #Create OptionParser Instance
                        usage= usage_msg)
parser.add_option("-n", "--numlines",               #Start defining options. Action "store" tells optparse
    action="store", dest="numlines",                #to take next argument and store to the right destination
    default=1,                                      #which is "numlines". Set the default value of "numlines"
    help="output NUMLINES lines (default 1)")       #to 1 and help message

options, args = parser.parse_args(sys.argv[1:])     #Options: an object containing all options arg
                                                    #Args: List of positional args leftover after parsing


try:                                                #Try Block
    numlines = int(options.numlines)                #Get numline from options and convert to int
except:                                             #Exception Handling
    parser.error("invalid NUMLINES: {0}"            #Error message if not int type, replace {0} with input
                .format( options.numlines ))

if numlines < 0:                                    #If numlines is negative, error messsage
   parser.error("negative count: {0}"
        .format( numlines ))

if len(args) != 1:                                  #If length of args is not 1, then error message
    parser.error("wrong number of operands")

input_file = args [0]                               #Assign first and only arg to variable input_file

try:                                                #Try Block
    generator = randline(input_file)                #Instantiate randline object with input_file
    for index in range(numlines):                   #For loop from 0 to numlines - 1
        sys.stdout.write(generator.chooseline())    #Print randomly chosen line
except IOError as (errno, strerror):                #Exception handling
    parser.error("I/O error({0}): {1}".             #Error message in the format "I/O error (errno):strerror"
        format( errno , strerror ))

if __name__ == "__main__":                      #Make the Python file a standalone program
    main()
```

## Considerations for shuf.py

- ArgParse
  - OptParse is deprecated in Python3 so use argParse module instead
  - https://docs.python.org/3.8/library/argparse.html#module-argparse
- Debugging
  - There are Python debuggers you can use, but also feel free to use Python's print statement. It will print out details of any object provided
- Shuf
  - Try the original shuf utility to compare behaviors
    - https://www.gnu.org/software/coreutils/manual/html_node/shuf-invocation.html
  - Check the source code in C for ideas
    - https://github.com/coreutils/coreutils/blob/master/src/shuf.c