SMU | Lyle School of Engineering

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Feb 27 2020, Lecture 12

Agenda

- Agenda:
  - Mid term exam on March 24 (Tuesday)
  - Object Oriented Programming
  - Data structure and Algorithms

Public, private and protected access modifiers

- Classical Objected-Oriented languages, such as C++ and Java controls the access to class resources by public, private and protected keywords
  - Public - accessible from outside the class
  - Private - they can be handled only from within the class
  - Protected - accessible from within the class and also available to its child classes

Public, private and protected access modifiers

- Python doesn't have any mechanism that effectively restricts access to any instance variable or method
- All members in a Python are public by default. Any member can be accessed from outside the class environment
- Python prescribes a convention of prefixing the name of the variable/method with single or double underscore to emulate the behavior of protected and private access specifiers

Public, private and protected access modifiers

- public by default

```python
class Employee:
    def __init__(self, name, sal):
        self.name = name
        self.salary = sal

e1 = Employee("carl", 10000)
print(e1.name)
```

access

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
carl

Process finished with exit code 0
```

- use one _ for protected

```python
class Employee:
    def __init__(self, name, sal):
        self._name = name    # protected attribute
        self._salary = sal   # protected attribute

e1 = Employee("carl", 10000)
print(e1._salary)
```

access

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
10000

Process finished with exit code 0
```

Public, private and protected access modifiers

- Use two underscore __ for private

```python
class Employee:
    def __init__(self, name, sal):
        self.__name = name    # private attribute
        self.__salary = sal   # private attribute

e1 = Employee("carl", 10000)
print(e1.__salary)
```

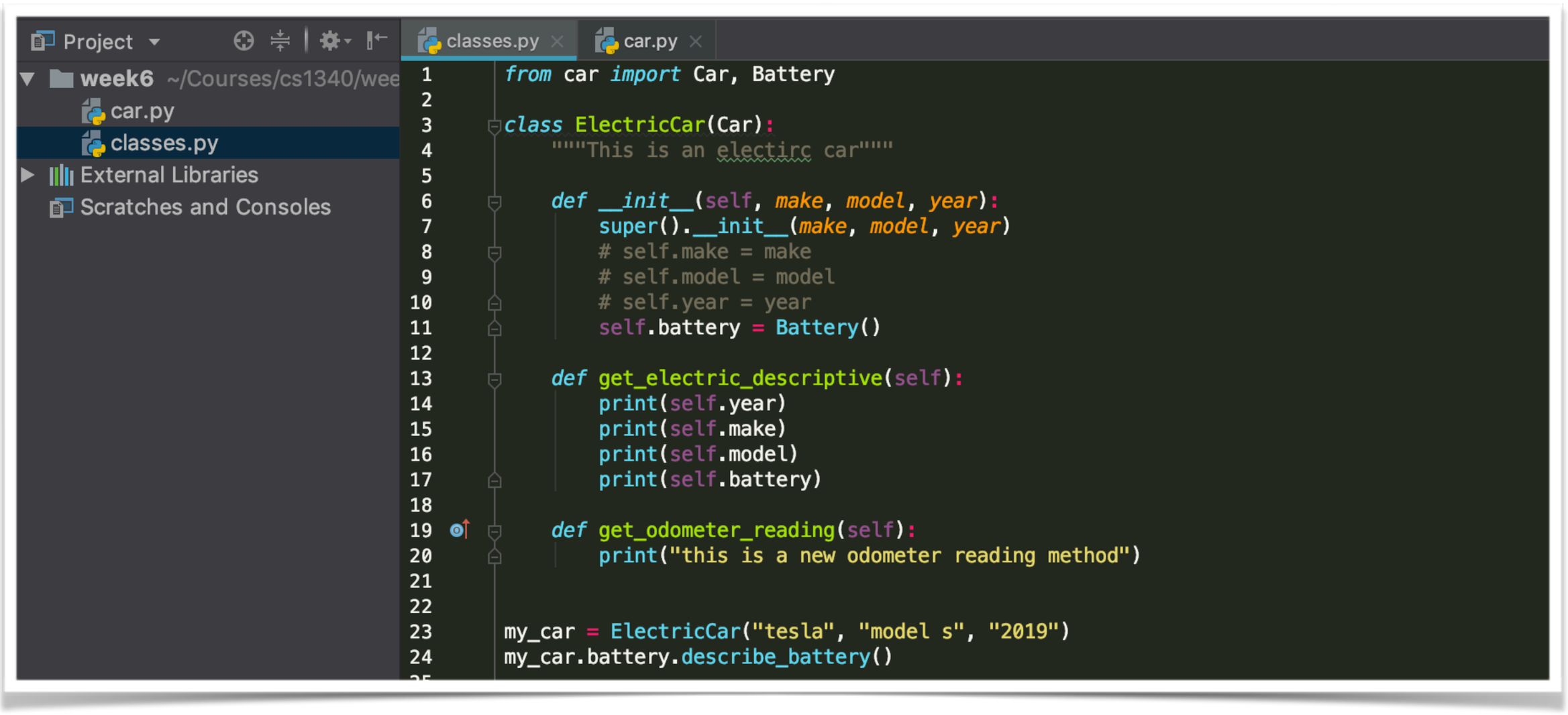
```
access
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/access.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week6/access.py", line 7, in <module>
    print(e1.__salary)
AttributeError: 'Employee' object has no attribute '__salary'

Process finished with exit code 1
```

- But if you are using Python 2. Python performs name mangling of private variables. Every member with double underscore will be changed to _object._class__variable. If so required, it can still be accessed from outside the class, but the practice should be refrained.

Demo

Importing Classes

- Store classes in modules and import the classes you need into your main program



The Python standard library

- The *Python standard library* is a set of modules included with every Python installation
- You can use any function or class in the standard library by including a simple import statement

```python
import datetime

# This class method creates a datetime object with the current date and time
now = datetime.datetime.today()

print(now.year)
print(now.hour)
print(now.minute)

long_ago = datetime.datetime(1999, 3, 14, 12, 30, 58)

print(long_ago)
print(long_ago < now)
```

std

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/std.py
2019
15
22
1999-03-14 12:30:58
True
```

The Python standard library

- math

```python
import math

# These are constant attributes, not functions

print(math.pi)
print(math.e)

# Round a float up or down
print(math.ceil(3.3))
print(math.floor(3.3))

# Natural logarithm
print(math.log(5))

# Square root
print(math.sqrt(10))

# Trigometric functions
print(math.sin(math.pi/2))
print(math.cos(0))

```

std

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/std.py
3.141592653589793
2.718281828459045
4
3
1.6094379124341003
3.1622776601683795
1.0
1.0
```

The Python standard library

- OrderedDict

```python
from collections import  OrderedDict

favorite_languages = OrderedDict()

favorite_languages["Alice"] = "Python"
favorite_languages["Sarah"] = "C"
favorite_languages["Jake"]  = "Ruby"
favorite_languages["Phil"]  = "Python"

for name, language in favorite_languages.items():
    print(name + "'s favorite language is " + language)

# print("\n")
```

favorite_language

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/favorite_language.py
Alice's favorite language is Python
Sarah's favorite language is C
Jake's favorite language is Ruby
Phil's favorite language is Python
```

The Python standard library

- Unordered Dict before Python 3.6

```python
# Since Python 3.6, dict are ordered
favorite_fruits = {}
favorite_fruits["Bob"] = "apple"
favorite_fruits["Ethan"] = "banana"
favorite_fruits["Alice"] = "orange"
favorite_fruits["Carl"]  = "pear"

for name, fruit in favorite_fruits.items():
    print(name + "'s favorite fruit is " + fruit)
```

favorite_language

```
/Users/xinyi/anaconda/envs/torch/bin/python /Users/xinyi/Courses/cs1340/week6/favorite_language.py
Carl's favorite fruit is pear
Bob's favorite fruit is apple
Alice's favorite fruit is orange
Ethan's favorite fruit is banana
```

Data Structure and Algorithm

- While trying to describe problems, we often realize they involve some kind of data we have to manipulate. It can be numbers, letters or something more complicated.
- Python comes with some predefined ways of representing data while writing our programs.
  - int
  - string
  - boolean
  - float

Data Structure and Algorithm

- Simple data types are great when dealing with individual values, but what happens when we need to perform operations on our data collectively?
- *Data Structures* allow us to group data together and describe the attributes and actions that can be performed on a particular instance of the data.
- Simple built in data structures in Python
  - list
  - tuple
  - dict

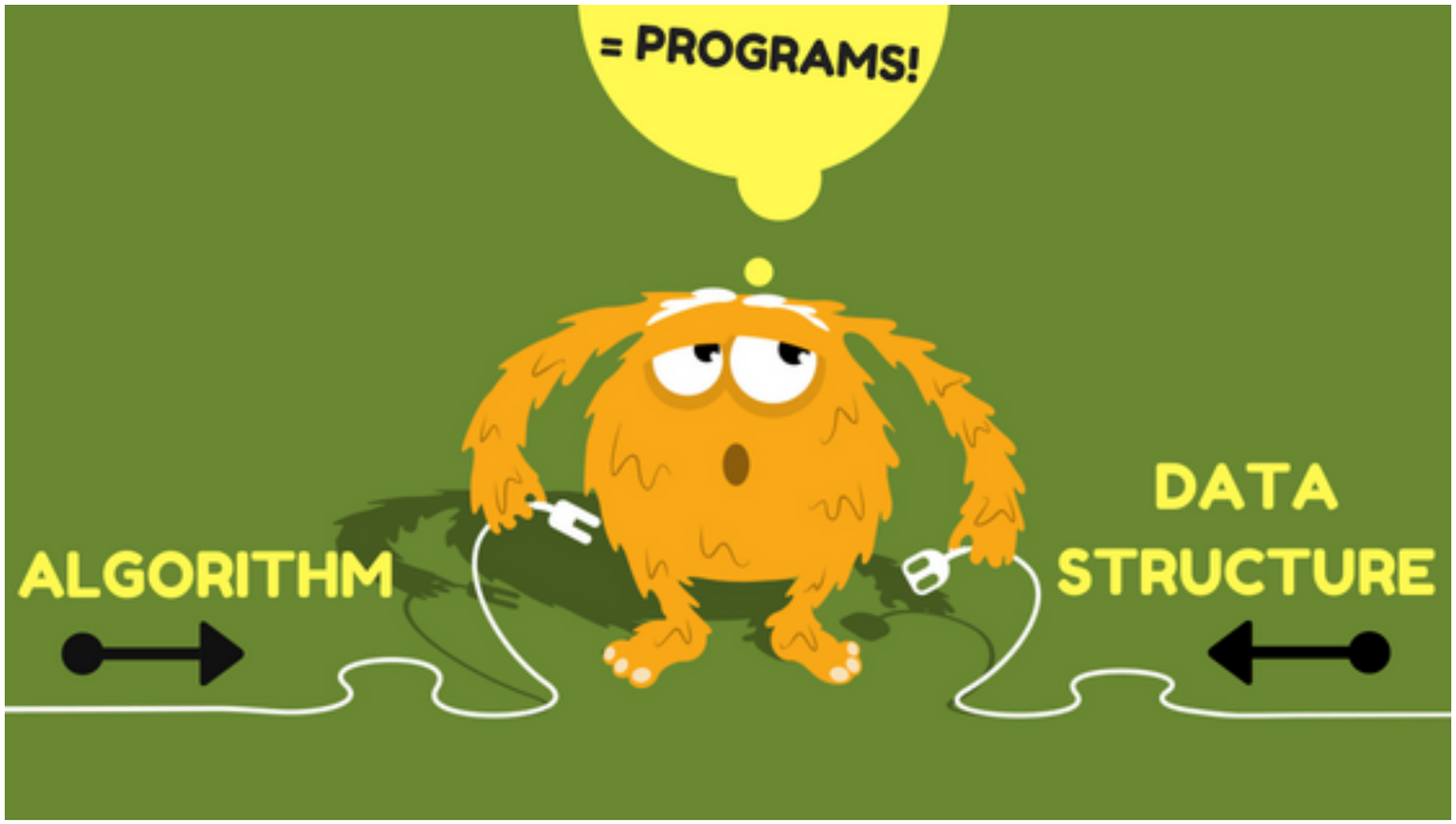
Data Structure and Algorithm

- Once we have modeled our real world data using *data structures*.
- We have to describe to the computer, step by step, instructions for solving a particular problem. This is often referred to as an *algorithm*.

https://medium.com/consonance/a-gentle-introduction-to-data-structures-and-algorithms-709cbb85855

Data Structure and Algorithm

- Programs = Data Structures + Algorithms



Data Structure and Algorithm

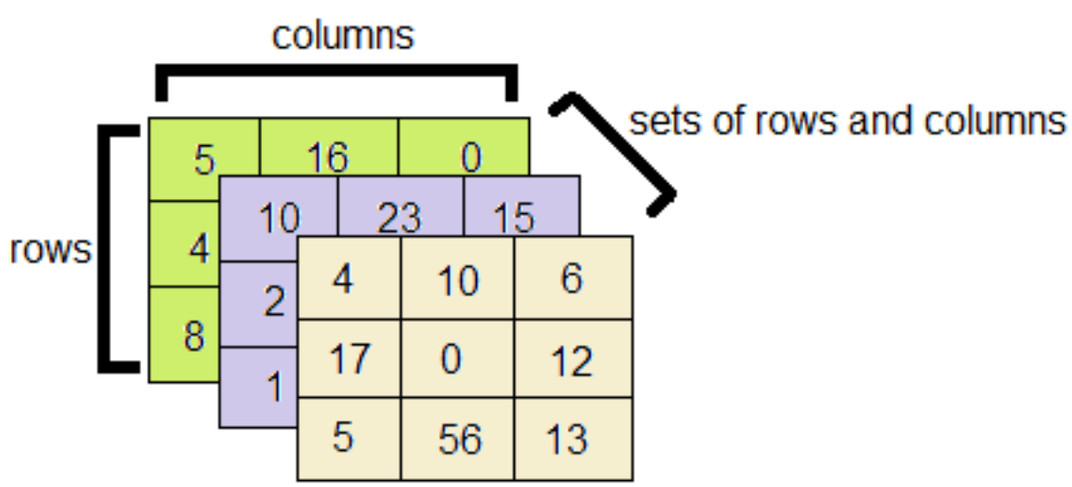
- Common data structures
  - Array (list in Python)
  - Linked-list
  - Queue
  - Stack
  - Trees
  - Graph
  - Hash-Tables
  - ...

Data Structure and Algorithm

- Multiple dimensional list
  - 2-D list

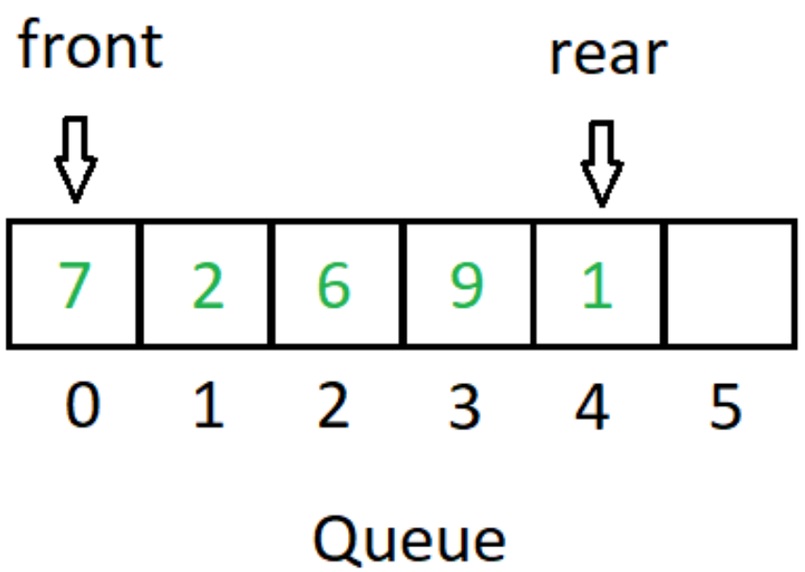
	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

  - 3-D list



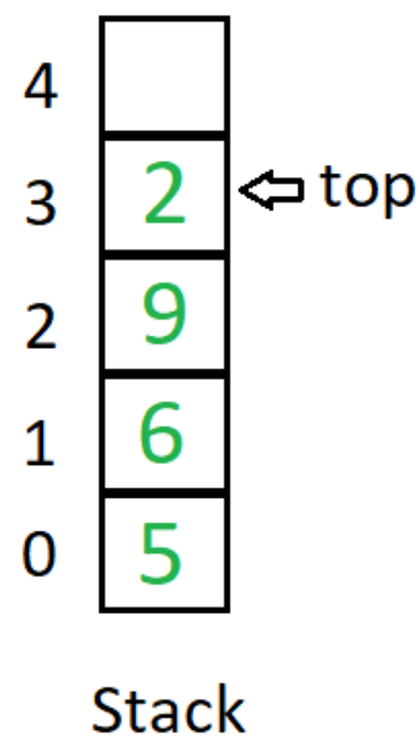
Data Structure and Algorithm

- Queue
  - first in first out (FIFO)
    - Handle orders
    - Ticket counter line where people who come first will get his ticket first



Data Structure and Algorithm

- Stack
  - first in last out (FILO)
    - An "**undo**" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.
    - **Back/Forward** stacks on browsers.



Data Structure and Algorithm

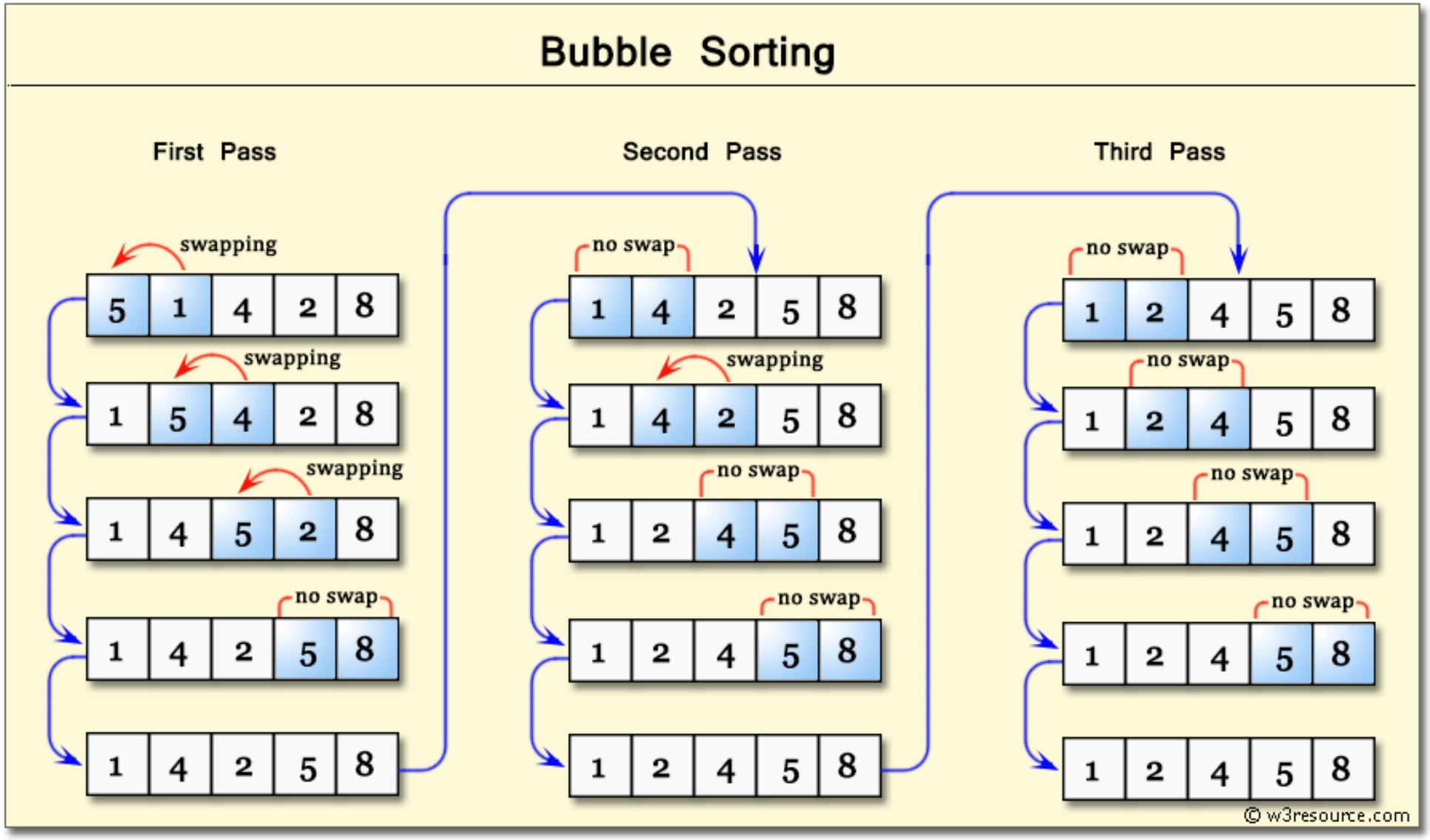
- Queues and Stacks in Python
  - Use list and pop()

Data Structure and Algorithm

- Common algorithms
  - Sort
    - Bubble sort
    - Selection sort
    - Quick sort
    - ...
  - Search
    - Binary search
    - Breadth First Search (BFS)
    - Depth First Search (DFS)
  - ...

Data Structure and Algorithm

- Example 1, Bubble sort



Data Structure and Algorithm

- Example 1, Bubble sort

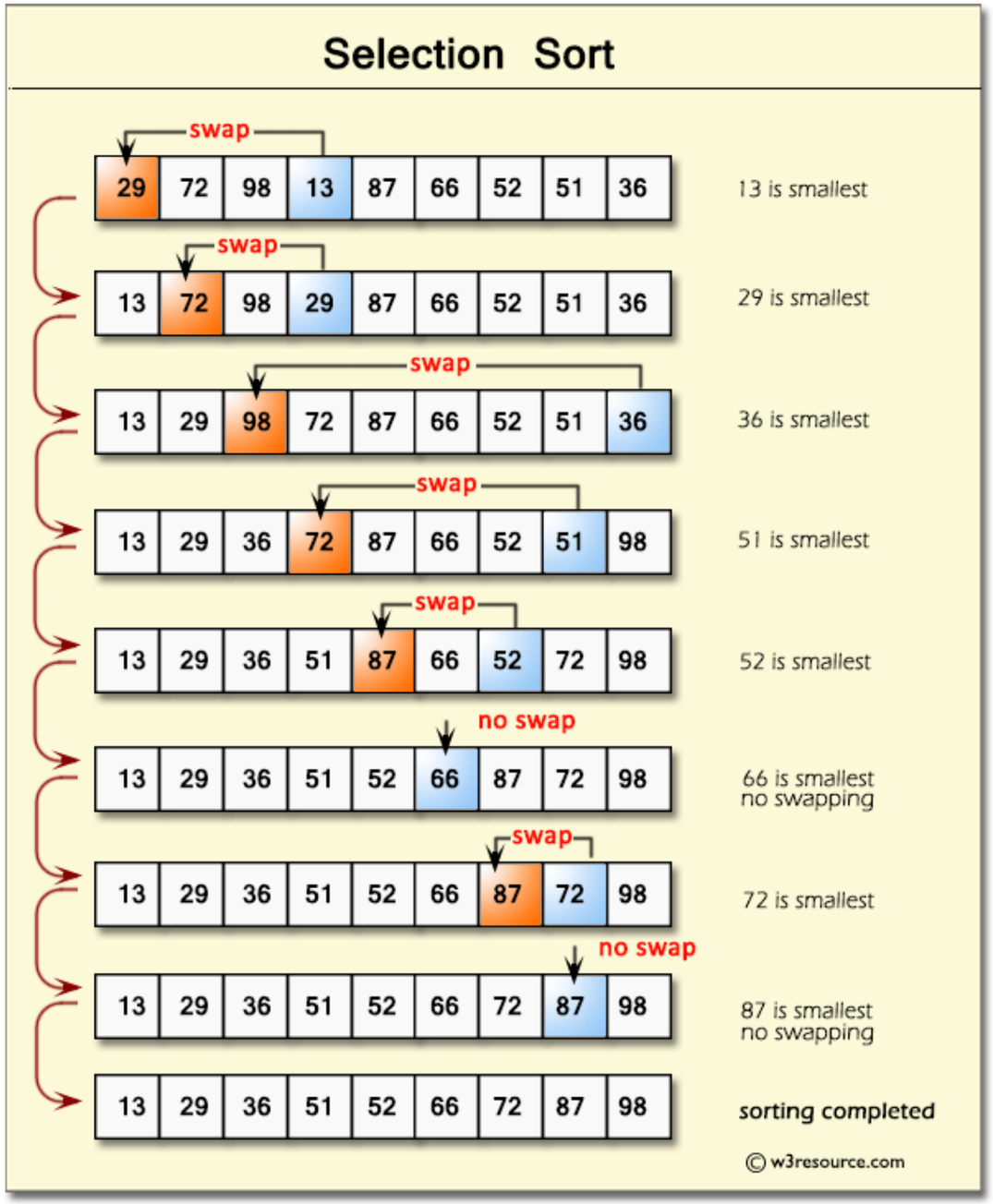
```python
# Bubble sort
def bubble_sort(nlist):
    for passnum in range(len(nlist)-1,0,-1):
        for i in range(passnum):
            if nlist[i]>nlist[i+1]:
                temp = nlist[i]
                nlist[i] = nlist[i+1]
                nlist[i+1] = temp

a_list = [34, 1, 2, 10, 12, 9]
bubble_sort(a_list)
print(a_list)
```

sort_alg

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/sort_alg.py
[1, 2, 9, 10, 12, 34]

Process finished with exit code 0
```

Data Structure and Algorithm

- Example 2, Selection sort



Data Structure and Algorithm

- Example 2, Selection sort

```python
# Selection sort
def selection_sort(nlist):
    for fillslot in range(0, len(nlist)):
        min_index= fillslot
        for location in range(fillslot, len(nlist)):
            if nlist[location] < nlist[min_index]:
                min_index = location

        temp = nlist[fillslot]
        nlist[fillslot] = nlist[min_index]
        nlist[min_index] = temp


a_list = [34, 1, 2, 10, 12, 9]
selection_sort(a_list)
print(a_list)
```

sort_alg

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/sort_alg.py
[1, 2, 9, 10, 12, 34]

Process finished with exit code 0
```

Data Structure and Algorithm

- Built in sort() in Python

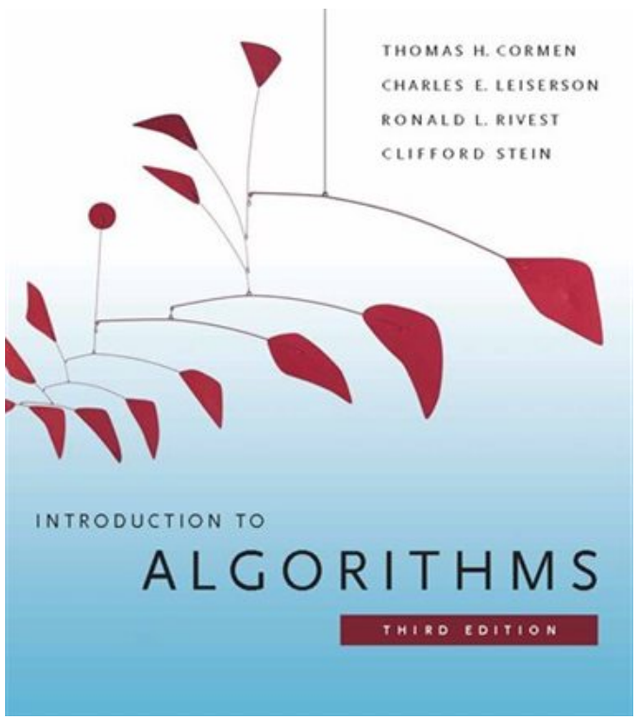
```python
a_list = [34, 1, 2, 10, 12, 9]
a_list.sort()
print(a_list)
```

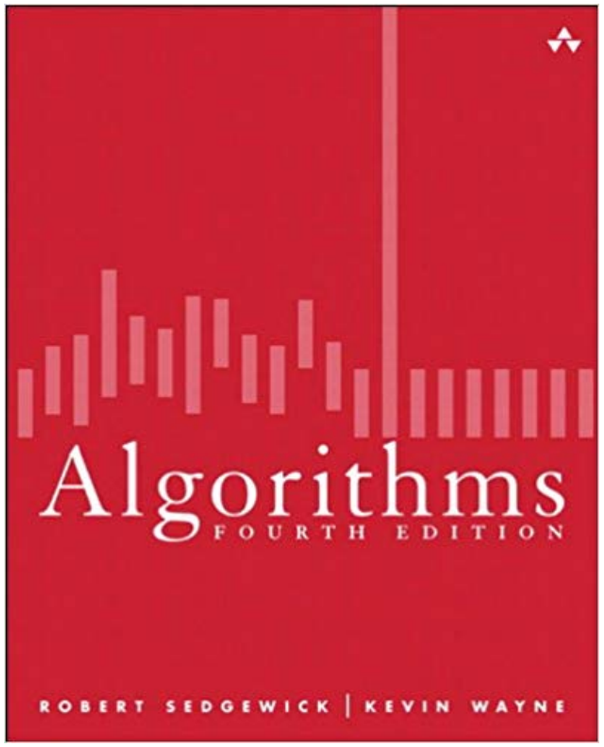
sort_alg

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week6/sort_alg.py
[1, 2, 9, 10, 12, 34]

Process finished with exit code 0
```

Data Structure and Algorithm

- Resources about data structures and algorithms





Next Week

- Jupyter notebook
- Data science
- numpy