- Questions

Object mystery

- ```
  class A():
      def __init__(self, x):
          self.x = x

      def bump(self, i):
          self.x += i

  class B():
      def __init__(self, x, y):
          self.x = x
          self.y = y

      def bounce(self, i):
          if i % 2 == 0:
              self.y += i
          self.x += i * 2

  x = 7
  a = A(x)
  a.bump(2)
  b = B(a.x, x)
  b.bounce(1)
  b.bounce(2)
  print(x, a.x, b.x, b.y)
  ```

Winter CS courses

- Department course guide: https://d31kydh6n6r5j5.cloudfront.net/uploads/sites/102/2019/08/cscourses1920.pdf

CS 201: Data Structures (111)

- required for the major
- The Data Structures course is all about how to store information intelligently and access it efficiently. How can Google take your query, compare it to billions of web pages, and return the answer in less than one second? How can one store information so as to balance the competing needs for fast data retrieval and fast data modification?
- Java
- Software development job interviews (including internships) mostly on material from this class
- prerequisite for everything
- Amy Csizmar Dalal, Anna Rafferty

CS 208: Computer Organization and Architecture (111)

- required for the major
- What is going on under there? How are positive and negative numbers represented, what actually happens when your code makes a function call, how is memory actually controlled
- C
- prereq for OS in the spring, people typically take 201 first, but not required
- Me!

CS 232: Art, Interactivity, and Robotics (111)

- In this hands-on studio centered course, we'll explore and create interactive three dimensional art. Using basic construction techniques, microprocessors, and programming, this class brings together computer science, sculpture, engineering, and aesthetic design. Collaborative labs and individual projects will culminate in a campus wide exhibition. No prior building experience is required.
- not offered very often
- David Musicant, Stephen Mohring (from ART)

CS 254: Computability and Complexity (111, 202/MATH 236)
  ```

- required for the major
- An introduction to the theory of computation. What problems can and cannot be solved efficiently by computers? What problems cannot be solved by computers, period?
- CS from a theoretical perspective
- Josh Davis

CS 252: Algorithms (201, 202/MATH 236)

- required for the major
- A course on techniques used in the design and analysis of efficient algorithms. We will cover several major algorithmic design paradigms (greedy algorithms, dynamic programming, divide and conquer, and network flow).
- CS from a theoretical perspective
- David Liben-Nowell

CS 257: Software Design (201)

- required for the major
- In this course, we will study techniques, tools, and habits that will improve your chances of writing good software. While working on several medium-sized programming projects, we will investigate code construction techniques, debugging and profiling tools, testing methodologies, UML, principles of object-oriented design, design patterns, and user interface design.
- Amy Csizmar Dalal

CS 312: Audio Programming (201)

- Students will learn the basics of MIDI and Digital Audio programming using C++. In the MIDI portion of the course, you'll learn to record, play, and transform MIDI data. During the Digital Audio portion of the course, you'll learn the basics of audio synthesis: oscillators, envelopes, filters, amplifiers, and FFT analysis. Weekly homework assignments, two quizzes, and two independent projects.
- John Ellinger

CS 330: Introduction to Real-Time Systems (201, 202/MATH 236)

- How can we prove that dynamic cruise control will brake quickly enough if traffic suddenly stops? How must a system coordinate processes to detect pedestrians and other vehicles to ensure fair sharing of computing resources? In real-time systems, we explore scheduling questions like these, which require provable guarantees of timing constraints for applications including autonomous vehicles. We will consider both theoretical and practical perspectives.
- Tanya Amert

CS 331: Computer Networks (201)

- The Internet is composed of a large number of heterogeneous, independently-operating computer networks that work together to transport all sorts of data to points all over the world. The fact that it does this so well given its complexity is a minor miracle. In this class, we'll study the structure of these individual networks and of the Internet, and figure out how this "magic" takes place.
- Amy Csizmar Dalal

CS 344: Human-Computer Interaction (201)

- The field of human-computer interaction addresses two fundamental questions: how do people interact with technology, and how can technology enhance the human experience? In this course, we will explore technology through the lens of the end user: how can we design effective, aesthetically pleasing technology, particularly user interfaces, to satisfy user needs and improve the human condition? How do people react to technology and learn to use technology? What are the social, societal, health, and ethical implications of technology?
- Sneha Narayan

Advanced objects

static variables

- the enemies in Space Invaders speed up the fewer of them there are (https://www.youtube.com/watch?v=MU4psw3ccUI)
- ```
  class Alien():
      count = 0

      def __init__(self):
          Alien.count += 1

      def die(self):
          Alien.count -= 1

  aliens = []
  print(Alien.count)
  for i in range(11):
      aliens.append(Alien())
  print(Alien.count)
  aliens[0].die()
  print(Alien.count)
  ```

inheritance
  ```

- ```
  class Person():
      def __init__(self, fname, lname):
          self.firstname = fname
          self.lastname = lname

      def fullname(self):
          return self.firstname + " " + self.lastname


  class Student(Person):
      def __init__(self, fname, lname, year):
          super().__init__(fname, lname)
          self.graduation_year = year

      def fullname(self):
          return self.firstname + " " + self.lastname + ", class of " + str(self.graduation_year)
  ```

Recursion

- Definiing something in terms of itself

recursively summing a list

- ```
  def sum(nums):
      if len(nums) == 0: # base case
          return 0
      return nums[0] + sum(nums[1:]) # recursive call
  ```

recursive count the number of elements in a list

- ```
  def length(xs):
      if xs == []: # base case
          return 0
      return 1 + length(xs[1:]) # recursive call
  ```
  ```