COMP_SCI 214

Welcome to Data Structures & Algorithms

PROF. SRUTI BHAGAVATULA

Overview

1. Course overview
2. Course staff
3. Course logistics
4. The language: DSSL2
5. Basics of DSSL2

~40 minutes

~40 minutes

Course overview

What if you need to write…

- A program that handles food orders
- A word processor that allows undoing of edits
- A calendar application which you can query to get appointments
- A contact tracing application
- A maps application that keeps track of locations and distances

- How can we write those complex programs?

In comes data structures

- All of these require your program to keep track of some **data**
  - Updating data
  - Getting data
- Data needs to be represented in some way to enable this
  - This “way” is called a data structure

Data structures are foundational

Bad programmers worry about the code. Good programmers worry about data structures and their relationship.

— Linus Torvalds

Data structures are fundamental when writing any but the most trivial of programs.

What is a data structure?

- A scheme for organizing data, to use it efficiently
- Two parts:
  - **Representation:**
    - Conceptual pieces of data to concrete building blocks
  - **Operations:**
    - How a client accesses and manipulates these conceptual pieces

Goals of any data structure

- **Correctness:**
  - Operations have the behavior they promised
  - Why? So our programs can rely on them!

- **Efficiency:**
  - Time: operations should be "fast"
  - Space: representations should not use too much memory

Now, what if you need to…

- Sort a list of names
- Search for a name in a list easily
- Look up a word in a digital dictionary
- Find the shortest route between two locations
- Identify all links on a page, all links at those links, and so on

In comes algorithms

- Need a way to do these things efficiently
- Data structures and algorithms go hand in hand
- Algorithms work on one or more data structures
  - They use operations of data structures
- Efficient data structure representations enable efficient algorithms

Goals of this course

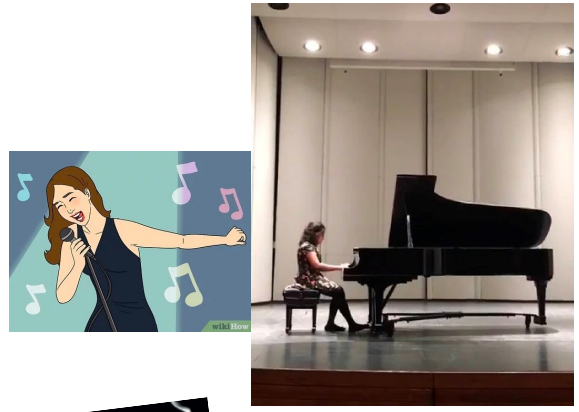
- Show you what kinds of data structures are out there
  - Not exhaustive but aims to cover a lot of ground
  - So you have a large **"vocabulary"** when programming
  - And can make informed choices about how to represent your data
- Introduce you to important algorithms that operate on data structures
- Gets you thinking like a **computer scientist** not a programmer
- Gets you to focus on the **bigger picture** of programming (necessary for 300+ level)
  - Interfaces vs. implementations
  - Invariants
  - Thinking beyond functional correctness
  - I.e., writing programs that others can **build on and rely on**

Course staff

Instructor

- **Who:** Prof. Sruti Bhagavatula
  - "Shru-thee Bhaa-guh-vuh-thoo-laa"
- **What to call me:** Prof. Sruti, Prof. Bhagavatula, Prof./Professor
- **My pronouns:** she/her/hers
- **My research:** security and privacy

My hobbies



Teaching Assistants (TAs) and Peer Mentors (PMs)

- **Graduate TAs (2):**
  - Leif Rasmussen (PhD student)
  - Tochukwu Eze (PhD student)
- **PMs (14):**
  - Alison Bai, Kate Bryar, Alina Chen, Natalie Hill, Blanka Jarmoszko, Bennett Lindberg, Miya Liu, Ethan Piñeda, Mattie Poelsterl, Omar Sharaf, Srikrishna Sorna, Donald Texeria, Francis Velasco, Anthony Zheng

How to contact us

- Piazza post just to “Instructors”
- Piazza post to only me (for sensitive matters)
- Email only in emergencies (otherwise expect delays) or when needed

- However, for most matters, you’ll post publicly to Piazza (coming up)

Course logistics

Course philosophy: Learning comes first

- Last stop before 300-level classes/internships
- Custom grading system to emphasize learning over grades
  - Mainly qualitative, no weighted averages or curves
  - Details in syllabus: **please read in full!!**
  - **Goals:** reduce stress, enable you to produce your best work

Course philosophy: Low stakes

- Room to learn from your mistakes (wherever possible)
  - Programming assignments: feedback + resubmissions
- Bounded impact (for everything else)
  - You won't fail the class just for bombing one exam
- Flexibility for everyone, built-in
  - No questions asked, no need to ask
  - Late tokens, multiple paths to success, etc.

Course philosophy: High expectations

- We expect you to **consistently** produce **excellent** work
- One doesn't learn much by producing so-so work
  - Do that too much -> you barely learn and you fail the class

- **Heads up:** Our bar for high grades is intentionally **very high**
  - We want you to produce your **best** work!

Course philosophy: No partial credit

- In other classes: half-broken or insufficient work is rewarded with partial credit
  - But you are being rewarded for the wrong thing
  - You missed out on a lot of learning
  - Moving on and learning from your mistakes is a missed opportunity!
- Instead, **resubmissions!**
- Instead, half-broken work -> no credit, go fix it!
  - You still have concepts / good habits to learn
  - You'll be rewarded for learning from your mistakes and trying again with a resubmission

Course philosophy: Transparency

- Grading mechanisms, high-level expectations, etc. stated up front
  - Everything is in the syllabus and assignment handouts
- No surprises:
  - If you're headed for trouble, you can see it coming
    - Just follow the tables in the syllabus
    - Homework 1 will give you a tool to track your progress
- You're in control:
  - Grade determined entirely by your own work
  - Not by arbitrary measures (percentage cutoffs, curves, etc.)

Course philosophy

- We're confident all of you can succeed!!
  - Every one of you
  - We're here to help!
  - But you'll have to work for it, and be diligent!

- How can you succeed in this class? →

How can you succeed in this class?

- **Today:** read the syllabus in full
  - Avoid surprises down the road
- **Before lecture:** read supplemental materials, skim slides
  - Get an idea of what to expect and be ready with questions
- **During lecture:** pay attention and ask questions!
  - Either by raising your hand or anonymously on Piazza
  - If you can't make it to class, watch recording ASAP
- **After lecture:** review what we have seen
  - If something is not clear, ask on Piazza or in office hours
- **Start assignments early:** they'll get big
  - First one will be out today! Due next Thursday

Communication channels: Lectures

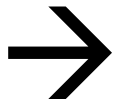
- Ask questions in class
- Raise your hand
- Post your questions on the Piazza thread for the day:
  - Our course staff will ask me the question on your behalf
  - Or they will answer it
  - Or I will answer it on Piazza after class
- Engage in discussions with your classmates for activities:
  - Graded in-class exercises (most lectures)
  - Other activities

In-class exercises

- During most lectures (excluding today)
- Short activity to make you critically think about and engage with lecture material
- Encouraged to discuss these with your classmates
- Graded on evidence of effort, meaningful justification, and often some degree of correctness
- See syllabus for how this fits into your overall grade -- modifiers

Communication channels: Canvas

- Syllabus (read it today!!)
- Up-to-date schedule
- Materials (slides, textbook, supplementary videos, etc.)
- Assignments (handouts and submission)
- In-class exercise links
- Lecture recordings
- Don't send messages to course staff there; instead use



Communication channels: Piazza

- Asynchronous Q&A about course content → usually fast responses!
- **Contact staff through Piazza** (for non-content related inquiries):
  - Make a private note to "Instructors" (or just me for sensitive matters)
  - **I prefer Piazza over email!** I may miss your email; expect delays ☹
  - We'll aim to respond within 2 weekdays
- Announcements will be posted there
- We encourage you to answer each other's questions!
  - But do not post elements of your solution
- **Read "Piazza guidelines" in syllabus!**

Communication channels: Office hours

- See Canvas for schedule and details → **they start today!!**
  - Some in-person, some online office hours
  - My office hours, PM office hours, TA office hours
  - Sign up in queue → PMs will come to you

Additional help

- Piazza and office hours should be the first stop for help
  - Army of course staff and abundant OH slots
- You can request 1:1 OH appointments or more help by
  - Posting to just me on Piazza → handled centrally
  - Please don't ask individual PMs to meet with you -> extra work will fall inequitably to those PMs

Assessments

- Programming assignments
- Self-evaluations for programming assignments
- Two online worksheet assignments
- Two exams (see Canvas for dates)
- In-class exercises
  - Most lectures (excluding today and exam days)
  - Graded on correctness, justification, or both
- Final project (program and design documents)

Classroom etiquette

- Learning (and teaching) this material requires focus and concentration
- Classroom environment should be conducive to that
  - Goal: minimize distraction and disruptions for us all
- Laptops in class are okay
  - If you're not working on what's going on in class, please sit further back to avoid being a distraction
- Please do not have side conversations during class
  - Distracting to myself and to your classmates
  - If you have a question, ask it to us (via the two channels)!
- Coming in late/leaving early?
  - Please sit further back, and just come in/leave quietly

Academic integrity

- Collaboration is usually good; plagiarism is bad
  - Know the line between them; be far away from that line
- We take academic integrity in 214 very seriously and so should you
- Consequences:
  - Biggest one: you won't learn → will definitely hurt you long-term
  - You'll be reported to the dean → no credit, F grade, suspensions, expulsions
  - 16 violation cases last spring → several resulting in failure, one in suspension

What counts as plagiarism?

- You should never see, share, describe, be described to, or be in the presence of a solution that is not yours from this quarter
  - Not your peers', not your friends', not something online from this quarter or a previous quarter
  - Not your old submissions from a previous quarter → you must delete all your code now
- AI tools are strictly forbidden → easiest way to not learn and not be competitive for jobs
- When in doubt, ask the instructor **before** you do something
- See the syllabus for a comprehensive list of **what is allowed** and **what is not allowed**

If you are struggling or just want to talk

- Reach out to me and I'm happy to meet with you
- My main job is to help you all succeed

- Whether you're struggling with
  - Course materials
  - Other stuff going on in your life
  - Other things

Pause

- Any questions?

The language

The DSSL2 language

- Stands for "Data Structures Student Language (version 2)"
- Close cousin of Python
  - If you know Python, learning DSSL2 should be easy
  - And vice versa

Why DSSL2?

- …and not just Python itself?

- Python doesn't have basic building blocks (basic arrays)
- Already contains a lot of useful data structures → takes away your opportunity to learn

- DSSL2 allows learning and practice with data structures without the burden of language-specific features and headaches
  - Plus avoids miscellaneous setup/environment headaches

Language skills are transferrable

- This class is not a "programming" class; it is a "computer science" class
  - Objective is to learn concepts, not a language
  - Concepts are language-independent
- Can easily transfer your programs in this class to other appropriate languages
  - Get practice with transferring your programming skills
  - Important as you grow as a computer scientist and programmer

How to install DSSL2

- DSSL2 is built on top of Racket
  - But DSSL2 != Racket! Very different languages.
- Install Racket 8.11 from racket-lang.org
  - Older versions will not work properly later on
- Follow instructions in DrRacket from syllabus and HW1
- Don't want to use DrRacket when programming?
  - See syllabus for alternatives

Basics of DSSL2

DSSL2 expressions and statements

```
3 + 5                                #comments start with #
6 * (3 + 5)


1 + 'hello'.len()                    # method call


let x = 5                            # slight difference from python
                                     # to avoid ambiguity (-> bugs)


if condition:                       # indentation matters! just like python
    do_some_stuff()
else:
    do_other_stuff(x, y, z)    # function call
```

DSSL2 functions

```
# hypotenuse(Number, Number) -> Number

# Finds the length of the hypotenuse given the lengths   # of the other two
sides of a right triangle.
def hypotenuse(a, b):
    if (a < 0 or b < 0):
        error('cannot have negative-length sides!')
    return (a * a + b * b).sqrt()

# ack(Natural, Natural) -> Natural
# Computes Ackermann numbers.
# Those will do a guest appearance later.
def ack(m, n):
    if   m == 0:     return n + 1
    elif n == 0:  return ack(m - 1, 1)
    else:         return ack(m - 1, ack(m, n - 1))
```

DSSL2 assertions and test cases

```
# assertions error and stop your program if they fail
# great for error checking and testing!
assert ack(0, 4) == 5

test 'arithmetic works':
    assert 2 * 5 == 10
    assert 2 + 5 == -1
    assert 2 - 5 == -3
```

```
# can give a name to tests
# can group assertions
# errors, then continues
# running after test block
```

In this class, we expect you to write your own tests. **See supplementary video on Canvas for advice.**

Why write tests?

Programming without tests is like running around with a blindfold and a blowtorch.

— Prem Seetharaman

DSSL2 programs

Every DSSL2 program starts with a #langline, followed by any number of statements:

```
#lang dssl2
let CM_PER_INCH = 2.54

# Converts centimeters to inches.
def cm_to_inch(cm):
    return cm / CM_PER_INCH

# Converts inches to centimeters.
def inch_to_cm(inches):
    return inches * CM_PER_INCH

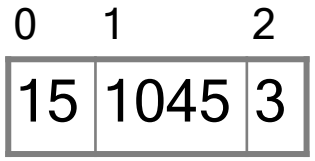
test 'round trip':
    assert inch_to_cm(cm_to_inch(17)) == 17 assert
    cm_to_inch(inch_to_cm(17)) == 17
```

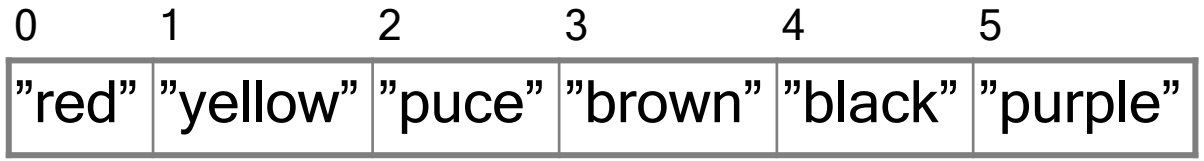
What are data structures made of?

- Conceptually: boxes and arrows
  - Boxes hold information
  - Arrows connect boxes to form larger structures
- Two kinds of boxes:
  - Vectors (or arrays)
  - Structs

Vectors

- Vectors contain sequences of data
  - *Indexed* by *integers* $0 \ldots n - 1$
  - Contents typically all of the same type
- Size fixed when *creating* the *specific vector*
  - A vector is created with size 10 → it has size 10 forever
  - But can create vector of different sizes
- Vectors take the same time to access an element…
  - Regardless of how far into the vector!
  - Regardless of how big the vector is!





Vectors in DSSL2

```
# Can construct vector literals.
let v = [ 0, 1, 1, 2, 4, 7, 13, 24, 44, 82 ]

test 'vector basics':
    assert v[3] == 2
    assert v[6] == 13
    assert v.len() == 10

test 'vector set':
    v[6] = 23
    assert v[6] == 23
```

Vectors in DSSL2

```
# Can construct vector literals.
let v = [ 0, 1, 1, 2, 4, 7, 13, 24, 44, 82 ]

test 'vector basics':  assert v[3] == 2
     assert v[6] == 13  assert v.len() == 10

test 'vector set': v[6] = 23
     assert v[6] == 23

# Vector comprehensions allow you to create a vector # using a "description"
rather than literal elements.  [ 0; 1000000 ]
# Creates a vector with `1000000` elements, all `0`s.  # Much nicer than
typing the whole thing!
# Supports more complicated descriptions too, see docs.
```

Example: `average`

```
# average (Vector<Number>) -> Number
# Averages the elements of a vector.
def average(vec):
    return sum(vec) / vec.len()

# sum(Vector<Number>) -> Number
# Sums the elements of a non-empty vector.
def sum(vec):
    let result = 0
    # for-each loop, like in python or C++
    # `v` becomes each element of the vector, in turn  # first iteration, `v`
    is `vec[0]`
    # second iteration, `v` is `vec[1]`  # etc.
    for v in vec:
        result = result + v
    return result
```

Iterating through a vector

- `let vec = [None; 10]`
- 3 ways to iterate through `vec`
  1. `for v in vec: # when you just need to access`

     `v`
  2. `for i in range(vec.len()): # when you need to update elements`

     `vec[i]`
  3. `for i, v in vec: # when you need to access and update`

     `v`

     `vec[i]`

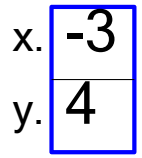
Pause

- Any questions?

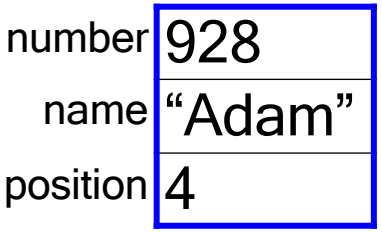
Structs

- Structs contain collections of data
  - *Accessed* by field *names*
  - Fields can be (and often are) of different types
- Fields determined when *defining* the struct *type*
  - Accessing any field takes the same amount of time

Positions have an xand a y field, both numbers.



Runners have a number field (integer), a namefield (string), and a position field (integer).



Structs in DSSL2

```
# Defines a new type of structure `posn` with fields
# `x` and `y`:
struct posn:
    let x
    let y

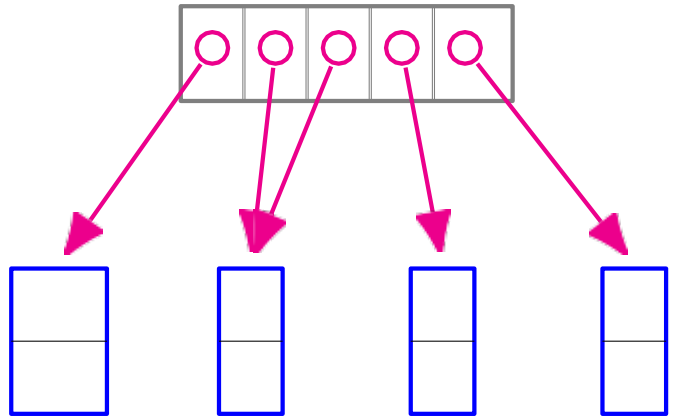
let p = posn(3, 4)

assert posn?(p)      # asserts that the result is true
assert p.x == 3
assert p.y == 4      # uses `.` notation like Python, etc.

p.x = 6
assert p.x == 6
assert p.y == 4
```

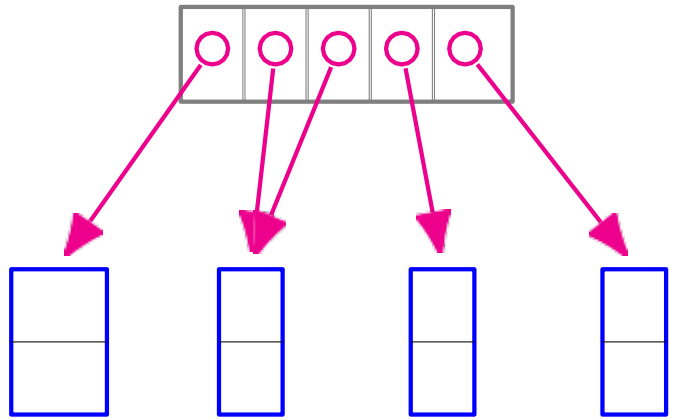
Arrows

- Arrows allow boxes to refer to one another
  - And thus to build arbitrarily-sized data
- Implicit in DSSL2 (and Python, etc.)
  - Struct in a struct field = actually an arrow to that struct!
  - Contrast pointers and references in C++ (*explicit* arrows)
  - You do not need to do anything special to get arrows in DSSL2, it's just for visual representation



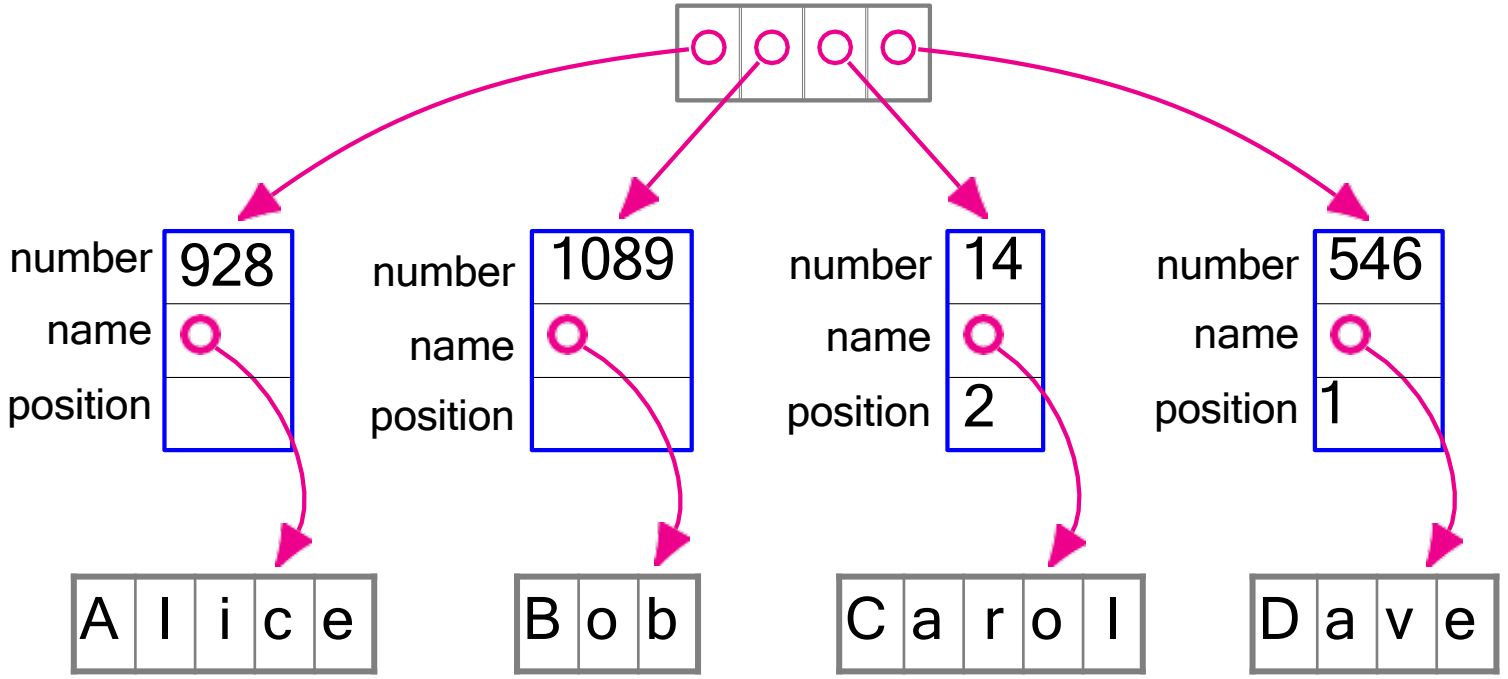
Arrows

- If two variables have an arrow to the same struct object, they point to same memory
  - Aliasing
  - See "Common DSSL2 errors" video on Canvas



Structs and vectors can be combined through arrows

- A vector of structs



Structs and vectors can be combined

- A vector of structs

```
struct runner:
    let number; let name; let position

let runners = [ runner( 928, "Alice", 4),
                runner(1089, "Bob",   3),
               runner(  14, "Carol", 2),
                runner( 546, "Dave",  1) ]
```

- **QUIZ**: Suppose we want to find out Carol's position:
  - **A:** runners[3].position
  - **B:** runners[2].position
  - **C:** runners.position[2]

Structs and vectors can be combined

- A vector of structs

```
struct runner:
    let number; let name; let position

let runners = [ runner( 928, "Alice", 4),
                runner(1089, "Bob",   3),
               runner(  14, "Carol", 2),
                runner( 546, "Dave",  1) ]
```

- **QUIZ**: Suppose we want to find out Carol's position:
  - **A:** runners[3].position
  - **B:** runners[2].position
  - **C:** runners.position[2]

You can also have a vector in a struct

- Try it out yourself!

Vectors vs. structs

- When should you use one versus the other?
- Do you have a **fixed** number of fields **you can name**?
  - Use a **struct**
- Does the number of elements **depend** on external factors and should the elements be of the same type?
  - Use an **array**

Classes

- Structs and vectors are enough to **represent** any data
- But data structures = representation + **operations**
  - Classes allow us to combine the two
- Classes ≈ structs with functions (methods)
  - A code organization mechanism to group data and operations together

Classes in DSSL2

```
class Posn:
    let x                                # fields: initialized by
    let y                                # the constructor

    def __init__(self, x, y):           # constructor: method
        self.x = x                       # with a special name
        self.y = y
    def get_x(self): self.x             # `return` is optional
    def get_y(self): self.y
    def distance(self, other):
        let dx = self.x - other.get_x()    # some other method
        let dy = self.y - other.get_y()    # fields are private to individual objects
        return (dx * dx + dy * dy).sqrt() # need to use getter for `other`
```

Classes in DSSL2

```
let p = Posn(3, 4)
assert p.get_x() == 3
assert p.get_y() == 4
assert_error p.x              # fields are
                              private

let q = Posn(0, 0)
assert p.distance(q) == 5
```

Contracts

- DSSL2 (like Python) does not have static types. Instead, you can use contracts to check values.
  - And catch mistakes before they snowball!
- ```
  def add_elt (x: int?, i: nat?, v: VecC[int?]) -> int?:
    return x + v[i]
  ```
  - ```
    # contract violation! expected `nat?`
    ```
  - ```
    assert_error add_elt(10, -5, [2, 3, 4] )
    ```
- We'll provide some contracts in assignments
- See supplementary video on Canvas (and docs) for details.
    ```

How can you learn DSSL2?

- **Learn largely through practice: trial and error**
- Practice implementing class code during or after class
- Homework 1 is meant to get you practice with DSSL2
  - And will help you track your progress in the class going forward
- **Supplementary videos on Canvas**
- Ask language-specific questions on Piazza
- Ask your questions in office hours
- You'll get a lot of errors and learn from them

For more DSSL2 information

- See the DSSL2 reference (or help desk).
- To search the help desk for DSSL2-specific topics (instead of every package under the sun):
  - Prefix your query with "T:dssl2"
  - For example, "T:dssl2 error" to search for DSSL2's error function
- Look at the DSSL2 documentation (https://docs.racket-lang.org/dssl2/)
  - Or search "dssl2 <something>" in Google
- Larger example: see recipes.rkt on Canvas

TODOs

- Read full syllabus today
- Look through supplementary materials and videos
- Read office hours guidelines
- Start homework 1
  - Office hours start today