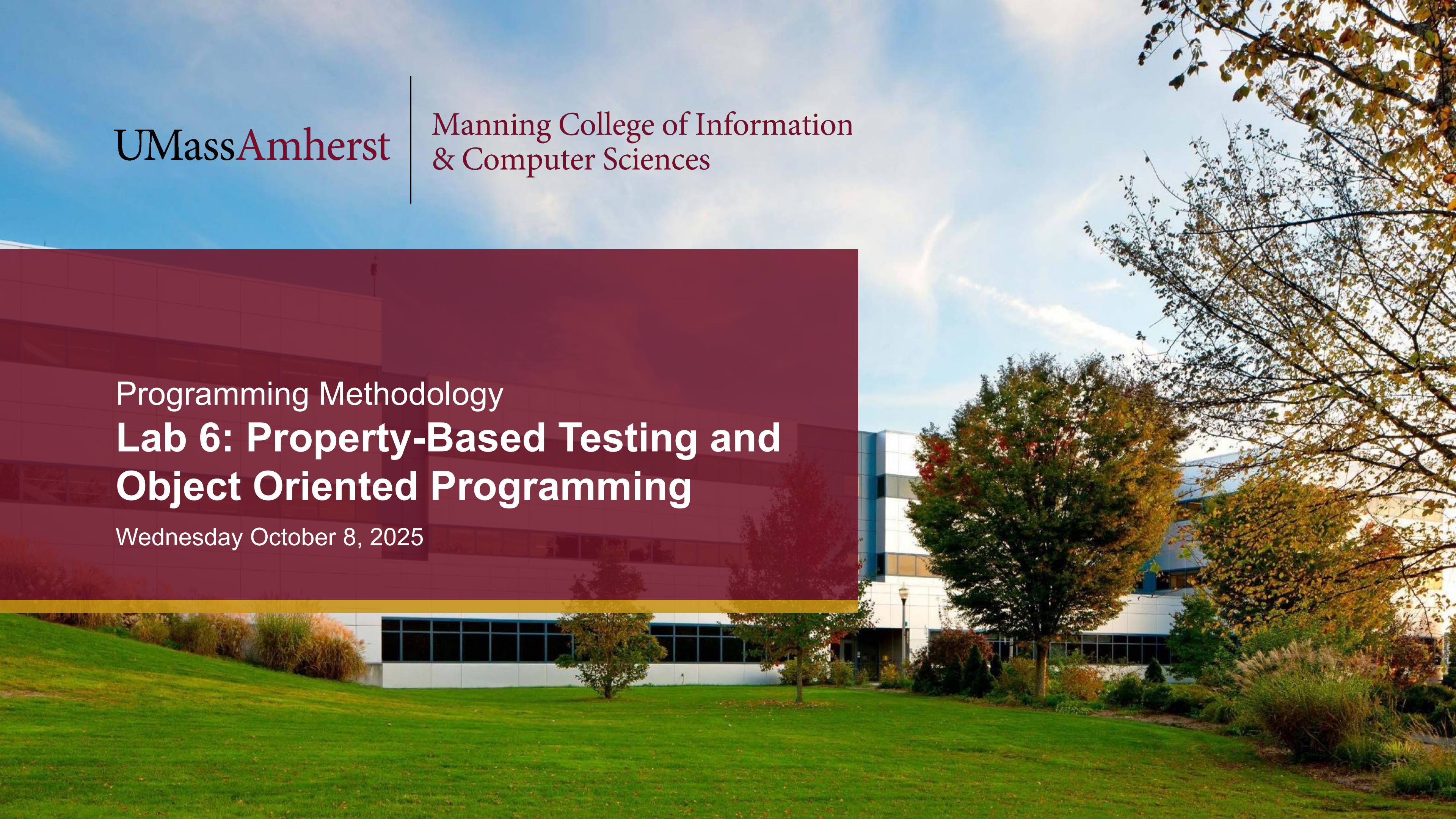


UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology

Lab 6: Property-Based Testing and Object Oriented Programming

Wednesday October 8, 2025

UMassAmherst | Manning College of Information & Computer Sciences

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
    - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

UMassAmherst | Manning College of Information & Computer Sciences

Reminders

- Homework 4a is due Sunday (10/12) at 11:59pm
    - Come to office hours for help!
- Homework 4b is due the following Sunday (10/19)
- Midterm 1 is being graded. Aiming to finish grading by Friday.
    - Exam solutions will also be posted later this week.
- Submit what you have at the end of lab to Gradescope. If you miss many submissions to Gradescope, we may penalize your lab grade.

Today's Goals

- Property-Based Testing
- Object Oriented Programming

Property-Based Testing

Used when a problem has **more than one** right answer.

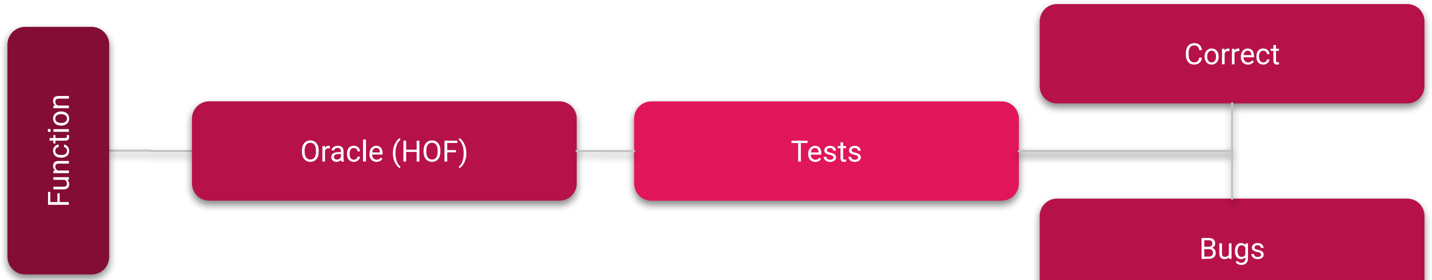
Steps to Property-Based Testing:

1. Start with a valid input to your problem
2. Run the algorithm on that input
3. Check that the result has all necessary characteristics

Oracle Functions

In this class, **oracles** are functions that determine whether or not a function correctly solves a given problem. We can determine this by performing property based testing on the outputs of the given function for various inputs.

For example, in homework 4a, you write an oracle for the Stable Matching Problem.



Exercise 1: Permutations

A function **genArray**(n: number): number[][] is supposed to generate an n × n array of numbers such that each row and column is a permutation of the numbers from 0 to n-1. Assume n is nonnegative.

Write an oracle that accepts the function genArray as input.

Use higher-order functions when appropriate. Try to write your implementation in O(n^2).

Solution 1: Permutations

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);

}
```

Solution 1: Permutations

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);

}
```

Solution 1: Permutations

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;

}
```

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {

      }

}
```

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {
            assert(p.length === n, ’length is n’);
      }

}
```

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
            assert(p.every(valid),'valid numbers');

      }

}
```

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
     let n = Math.floor(Math.random()*1000);
     let a = genArray(n);
     const valid = x => x % 1 === 0 && 0 <= x && x < n;
     function isPerm(p) {
          assert(p.length === n, 'length is n');
          assert(p.every(valid),'valid numbers');
          const f = Array(n).fill(1);
          assert(p.every(e => --f[e] === 0), 'no duplicates');
     }

}
```

Note: We're checking for duplicates in a single linear scan which takes O(n) time.
This approach works for n = 0 and an empty array.

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
            assert(p.every(valid),'valid numbers');
            const f = Array(n).fill(1);
            assert(p.every(e => --f[e] === 0), 'no duplicates');
      }
      assert(a.length === n, 'array has n rows');

}
```

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
            assert(p.every(valid),'valid numbers');
            const f = Array(n).fill(1);
            assert(p.every(e => --f[e] === 0), 'no duplicates');
      }
      assert(a.length === n, 'array has n rows');
      a.forEach(isPerm); // each row is a permutation

}
```

a is length n, and we call isPerm (an O(n) function) for each element of a. This will take $O(n^2)$ time

Solution 1: Permutations

UMassAmherst | Manning College of Information & Computer Sciences

Write an oracle that accepts the function genArray as input.

```
function oracle(genArray) {
      let n = Math.floor(Math.random()*1000);
      let a = genArray(n);
      const valid = x => x % 1 === 0 && 0 <= x && x < n;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
            assert(p.every(valid),'valid numbers');
            const f = Array(n).fill(1);
            assert(p.every(e => --f[e] === 0), 'no duplicates');
      }
      assert(a.length === n, 'array has n rows');
      a.forEach(isPerm); // each row is a permutation
      for (let k = 0; k < n; ++k) { isPerm(a.map(r => r[k])); } // each column is a permutation
}
```

Again, this will take O(n^2) time

Note: A different approach, where we check directly that every number from 0 to n – 1 is included requires a linear scan for each number. That would be O(n^2) per row/column and O(n^3) overall. Do not do this!

UMassAmherst | Manning College of Information & Computer Sciences

Review: Rectangle Class

In lecture, we explored drawing on a canvas as an example of classes and objects. For example, we looked at the Rect class, which represented a rectangle on a canvas.

Note that Rect was written in JavaScript, not TypeScript. **What changes do we need to make for TypeScript?**

```
class Rect {
    constructor(x, y, w, h) {
        this.loc = { x: x, y: y }; // top left corner
        this.width = w;
        this.height = h;
    }
    draw(ctx, color) {
        const x = this.loc.x;
        const y = this.loc.y;
        const w = this.width;
        const h = this.height;
        drawLine(ctx, x, y, x + w, y, color);
        drawLine(ctx, x + w, y, x + w, y + h, color);
        drawLine(ctx, x + w, y + h, x, y + h, color);
        drawLine(ctx, x, y + h, x, y, color);
    }
}
```

Review: Rectangle Class

In TypeScript, we must first declare class properties! We also need to add type annotations, as you are familiar with.

```
class Rect {
    loc: { x: number; y: number };
    width: number;
    height: number;

    constructor(x: number, y: number, w: number, h: number) {
        this.loc = { x: x, y: y };
        this.width = w;
        this.height = h;
    }

    // drawLine does not change
}
```

Exercise 2: OOP

UMassAmherst | Manning College of Information & Computer Sciences

Implement the **Circle** class. Similar to the Rect class, the Circle class should have a property **loc** representing the x, y position of the circle. It should also have a **constructor** and a **draw** method.

Hint: you may want to use the drawCircle function, which you can find in the lab.ts file.

```
class Circle {

  constructor(x: number, y: number, r: number) {
    // TODO
  }
  draw(ctx: SKRSContext2D, color: string) {
    // TODO
  }
}
```

Note: there is no autograder for this exercise.

Try testing your class by creating a Circle at x=80, y=70, and with radius 50. Then draw it using drawShape!

Solution 2: OOP

UMassAmherst | Manning College of Information & Computer Sciences

```
class Circle {
    loc: { x: number; y: number };
    radius: number;

    constructor(x: number, y: number, r: number) {
        this.loc = { x: x, y: y };
        this.radius = r;
    }
    draw(ctx: SKRSContext2D, color: string) {
        drawCircle(ctx, this.loc.x, this.loc.y, this.radius, color);
    }
}
```

We create a property **radius** to keep track of the circle’s radius. Along with the position **loc** of the center of the circle, this is enough for us to draw the circle.