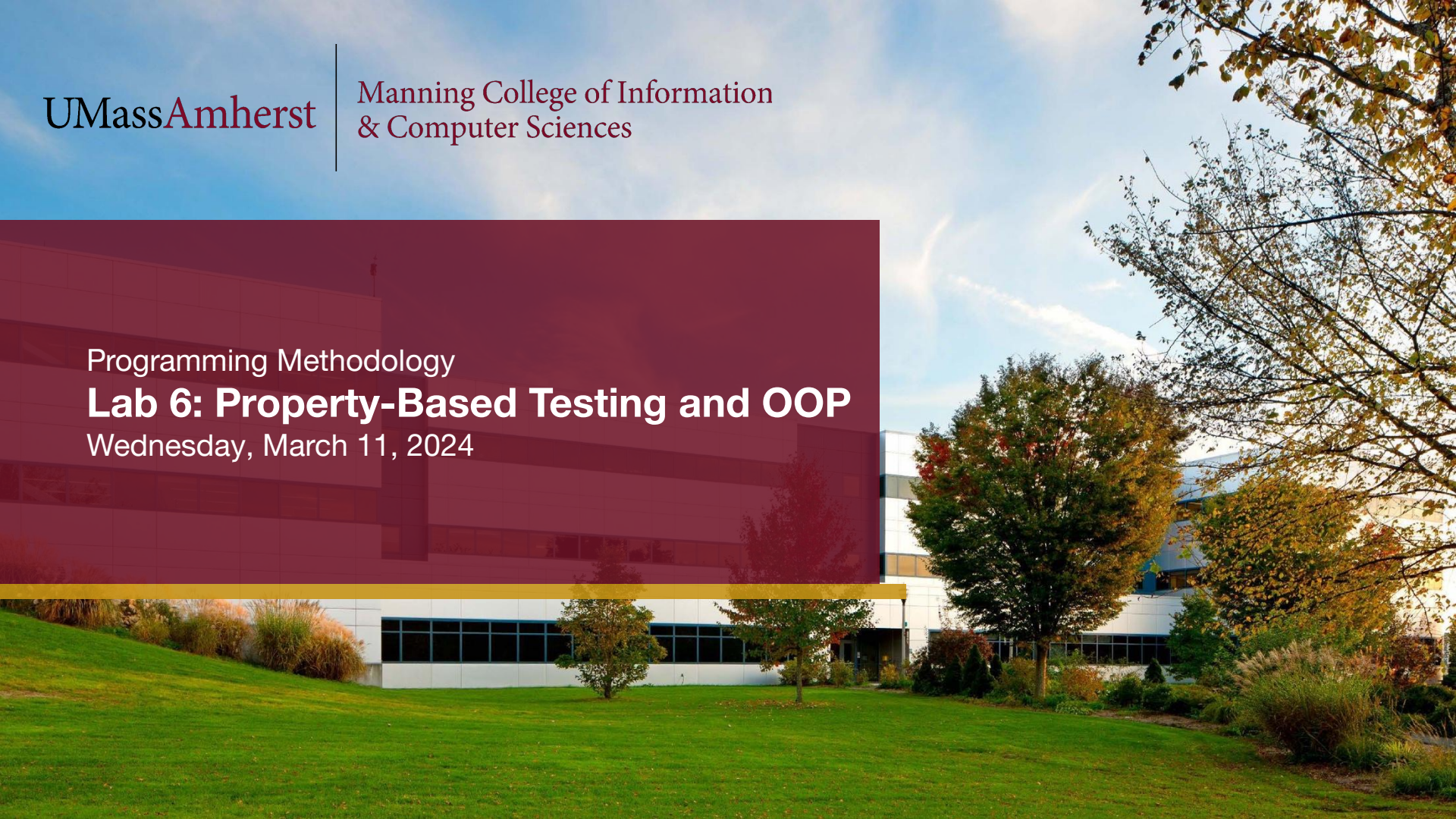


UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology

# Lab 6: Property-Based Testing and OOP

Wednesday, March 11, 2024

# Property-Based Testing



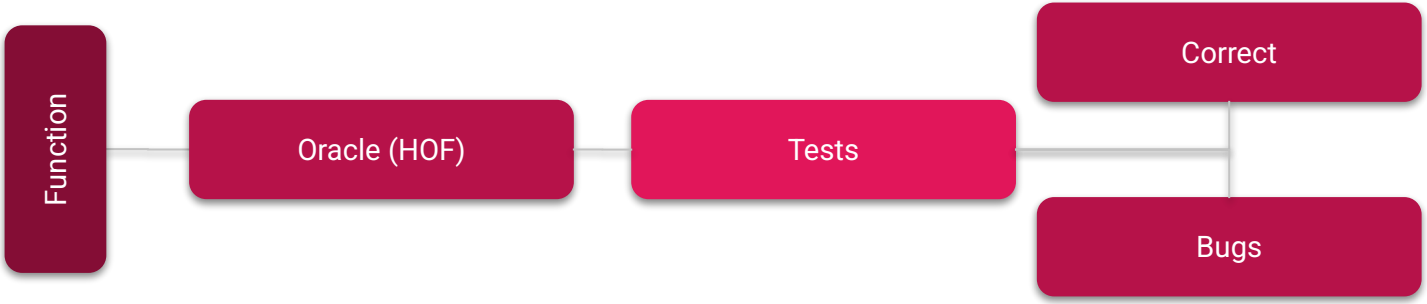
Used when a problem has **more than one** right answer

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





# Exercise: 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: Permutations



**Write an oracle that accepts the function genArray as input.**

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

}
```

# Solution: 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: 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: 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;
      function isPerm(p) {

      }

}
```

# Solution: 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;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
      }

}
```

# Solution: 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;
      function isPerm(p) {
            assert(p.length === n, 'length is n');
            assert(p.every(valid),'valid numbers');

      }

}
```

# Solution: 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;
      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: 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;
      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: 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;
      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: 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;
      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!

# Exercise: OOP



Think back to lecture where you discussed the shapes classes.
Before we start with this exercise, please familiarize yourself yourself for a few minutes with the code in the starter code. It should seem familiar to you.

Uncomment and run the three examples and make sure you understand!

- Your TA will demonstrate this.

Implement a **class** Translate whose constructor takes a shape and a change in x and change in y value. When the draw method is called with a CanvasRenderingContext2D (ctx) and a color, shift the canvas by dx and dy using ctx.translate, draw the shape on the moved canvas, and move the canvas back to the starting position.

Now run `npm run start`. You'll know your code is correct by the image.

# Exercise: OOP



```
class Translate {
  dx: number;
  dy: number;
  shape: Shape;

  constructor(shape: Shape, dx: number, dy: number) {
    this.dx = dx;
    this.dy = dy;
    this.shape = shape;
  }

  draw(ctx: CanvasRenderingContext2D, color: string) {
    ctx.translate(this.dx, this.dy);
    this.shape.draw(ctx, color);
    ctx.translate(-this.dx, -this.dy);
  }
}
```

Live code this! Show students how the images show up. :)

# Review Exam Questions



With the left over time we'll go over exam questions that you all found difficult.

Let us know which ones you want to go over!