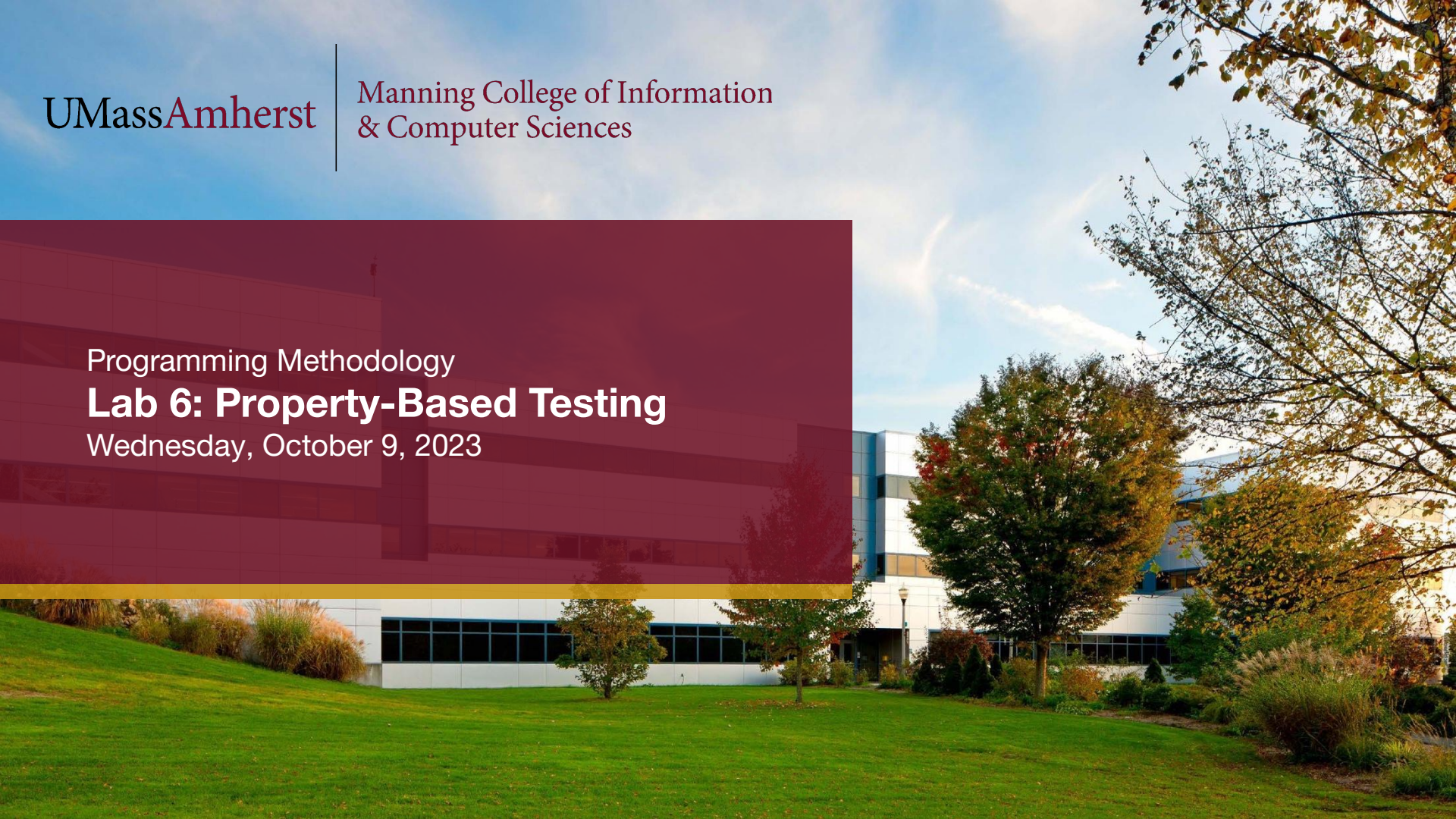UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology
Lab 6: Property-Based Testing
Wednesday, October 9, 2023

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

Property-Based Testing

UMassAmherst

Manning College of Information
& Computer Sciences

Ex: Given a set of numbers x = {1,2,...,N-1,N}, generate a random subset of x.
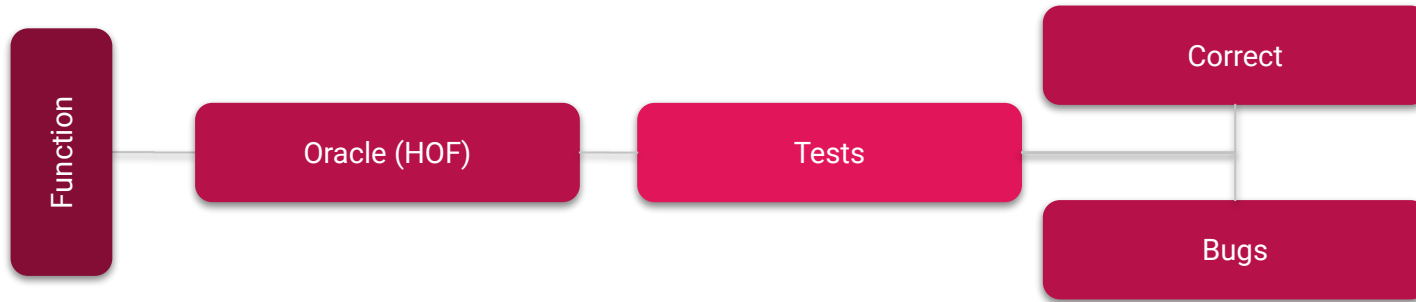**What are the properties of a correct output y to this problem?**

1. The number of elements in y must be less than or equal to N.
2. Every number in y must exist in x.
3. There must be no duplicates in y.

Oracle Functions

Function → Oracle (HOF) → Tests → Correct / Bugs

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {

}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;

}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){

    }
}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){
        let x = arrayFrom1ToN(N); // x is {1,2,...,N-1,N}

    }
}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){
        let x = arrayFrom1ToN(N); // x is {1,2,...,N-1,N}
        let y = subsetFunc(x);

    }
}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){
        let x = arrayFrom1ToN(N); // x is {1,2,...,N-1,N}
        let y = subsetFunc(x);
        test('length of y less than or equal to N', function(){
            assert(y.length <= N);
        })
    }
}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){
        let x = arrayFrom1ToN(N); // x is {1,2,...,N-1,N}
        let y = subsetFunc(x);
        test('length of y less than or equal to N', function(){
            assert(y.length <= N);
        })
        test('all elements in y are in x', function(){
            assert(y.every(d => x.includes(d)));
        })
    }
}
```

What would an oracle function for the subset function look like?

```
function oracle(subsetFunc) {
    let n = 100;
    for(let N = 0; N < n; ++N){
        let x = arrayFrom1ToN(N); // x is {1,2,...,N-1,N}
        let y = subsetFunc(x);
        test('length of y less than or equal to N', function(){
            assert(y.length <= N);
        })
        test('all elements in y are in x', function(){
            assert(y.every(d => x.includes(d));
        })
        test('y does not contain duplicates', function(){
            assert(y.every(d => (y.indexOf(d) === y.lastIndexOf(d))));
        })
    }
}
```

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).

Write an oracle that accepts the function genArray as input.

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

}
```

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;

}
```

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');
    }

}
```
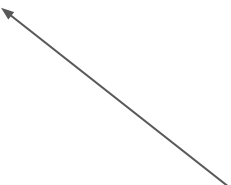
Write an oracle that accepts the function genArray as input.

```
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');

    }

}
```

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.

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');

}
```

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!

Review Exam Questions

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

Suggestion: Q4 or Q6, but we can go over any you want.

Write a function Q4 trackTemp(val: number): (diff: number) => [temp: number, days: number] that takes as argument the noon temperature on day 0 and returns a closure that can be called repeatedly. A call represents a new day, with a change in the noon temperature by an amount diff (positive, negative or zero) from the day before. The closure should return a pair of numbers: the noon temperature on the current day, and the number of consecutive days (≥ 1) that have passed without the temperature change diff having switched sign (zero is not a sign switch).

```
function trackTemp(val: number): (diff: number) => [temp: number, days: number] {
  let days = 0, lastS = 0;
  return diff => {
    if (lastS * diff < 0) days = 0;      // sign change
    if (diff !== 0) lastS = diff;        // try to make lastS <> 0, or keep it
    return [val+=diff, ++days];
  }
}
```

What are some relevant test scenarios?
Explain input, output, their purpose and what behavior they are testing.

An array represents employee wage increases in the years since their union was founded. There is one entry per year, representing the increase from the previous year as a ratio (assumed > 1). The first array entry is the increase in the year following the founding year. Write a function calcIncrease(byYear: number[]): ((wage: number) => number)[] that takes such an array and returns an array of closures, with the same length. When called with a wage amount for the founding year, the closure for each year will return the adjusted wage in that year, after all increases. Use reduce, no loops. Avoid needless repeated computations.

```
type numFun = (x: number) => number

function calcIncrease(byYear: number[]): numFun[] {
  const res: numFun[] = [];
  byYear.reduce((acc, rate) => {
    acc *= rate;
    res.push(x => x * acc);
    return acc;
  }, 1);
  return res;
}
```

Exercise 2: OOP

Define an iterator class over an array.

Define a class Polygon (defined by an array of points in 2D) which has a makeIterator() method for its vertices.

Extend the class to CenteredPolygon which has a method to compute its center.

The center of weight has the average of the vertex coordinates.

```
class ArrayIterator<T> implements IterableIterator<T> {.......}
```

```
class Polygon {.......}
```

```
class CenteredPolygon extends Polygon {.......}
```

UMassAmherst | Manning College of Information & Computer Sciences

```typescript
class ArrayIterator<T> implements IterableIterator<T> {
    private array: T[];
    private index = 0;
    constructor(private array: T[]) {this.array = array}
    public next(): IteratorResult<T> {
        if (this.index < this.array.length) {
            return this.array[this.index++];
        } else {
            return null;
        }
    }
}
```

```typescript
class Polygon {
    private vertices: [number,number][];
    constructor(private vertices:[number, number][]) {
            this.vertices = vertices;
        }

    public makeIterator(): IterableIterator<number[]>
        {
        return new ArrayIterator<number[]>(this.vertices);
    }
}
```

```typescript
class CenteredPolygon extends Polygon {
    public Center(): number[] {

        const numVertices = this.vertices.length;

        let x = 0;
        let y = 0;
        const iterator = this.makeIterator();
        let current = iterator.next();

        while (current !== null) {
            x += current[0];
            y += current[1];
            current = iterator.next();
        }


        return [x / numVertices, y / numVertices];
    }
}
```

Exercise 2: Solution 2

```typescript
interface Iterator<T> {
    next(): { value: T, done: boolean };
}
```

```typescript
class Polygon {
    private vertices:[number, number][];

    constructor(vertices:[number, number][]) {
      this.vertices = vertices;
    }

    makeIterator(): Iterator<number[]> {
      let index = 0;
      const vertices = this.vertices;

      return {
        next() {
          if (index < vertices.length) {
            return { value: vertices[index++], done: false };
          } else {
            return { value: null, done: true };
        }}};}}
```

```typescript
class CenteredPolygon extends Polygon {
    private readonly sides: number;

    constructor(vertices: number[][]) {
      super(vertices);
      this.sides = vertices.length;
    }
    const numVertices = this.vertices.length;
    center(): number[] {
      let x = 0;
      let y = 0;
      const iterator = this.makeIterator();
      let current = iterator.next().value;

      while (current !== null) {
        x += current[0];
        y += current[1];
        current = iterator.next().value;
      }

      return [x / this.sides, y / this.sides];
    }
  }
```

UMassAmherst | Manning College of Information & Computer Sciences

```typescript
class ArrayIterator<T> {
  private arr: T[];
  private idx: number;
  constructor(a: T[]) { this.arr = a; this.idx = -1;
}
  next(): T | null {
    return ++this.idx < this.arr.length ?
                    this.arr[this.idx] : null;
  }
}
```

```typescript
class Polygon {
  private vertices: [number,number][];
  constructor(vertices: [number,number][]) {
    this.vertices = vertices;
  }
  makeIterator() {
    return new ArrayIterator(this.vertices);
  }
}
```

```typescript
class CenteredPolygon extends Polygon {
  center() {
    const it = this.makeIterator();
    let xs = 0, ys = 0, cnt = 0;
    for (let v: [number, number] | null;
            (v = it.next()) != null; ++cnt) {
      xs += v[0]; ys += v[1];
    }
    return [xs/cnt, ys/cnt];
  }
}
```