Class 2: Testing

6.102 — Software Construction
Spring 2024

What does this code do?

- Python and TypeScript behave the same way here
- it helps to draw a diagram

```
s = "a"
t = s
s += "b"
print(s,t)

s.upper()
print(s,t)

v = []
v.append(s)
w = v
w.append(t)
print(v,w)
```

```
let s = "a";
let t = s;
s += "b";
console.log(s,t);

s.toUpperCase();
console.log(s,t);

let v = [];
v.push(s);
let w = v;
w.push(t);
console.log(v,w);
```

Exercise: yellkey.com/**grow**

Nanoquiz: yellkey.com/**consider**

Clicker: `clicker.mit.edu/6.102`

What does this code do?

- Python and TypeScript behave the same way here
- it helps to draw a diagram

```python
s = "a"
t = s
s += "b"
print(s,t)

s.upper()
print(s,t)

v = []
v.append(s)
w = v
w.append(t)
print(v,w)
```

```typescript
let s = "a";
let t = s;
s += "b";
console.log(s,t);

s.toUpperCase();
console.log(s,t);

let v = [];
v.push(s);
let w = v;
w.push(t);
console.log(v,w);
```

Exercise: yellkey.com/**grow**

Nanoquiz: yellkey.com/**consider**

Clicker: `clicker.mit.edu/6.102`

What does this code do?

- Python and TypeScript behave the same way here
- it helps to draw a diagram

```python
s = "a"
t = s
s += "b"
print(s,t)

s.upper()
print(s,t)

v = []
v.append(s)
w = v
w.append(t)
print(v,w)
```

```typescript
let s = "a";
let t = s;
s += "b";
console.log(s,t);

s.toUpperCase();
console.log(s,t);

let v = [];
v.push(s);
let w = v;
w.push(t);
console.log(v,w);
```

Exercise: yellkey.com/**grow**

Nanoquiz: yellkey.com/**consider**

Clicker: `clicker.mit.edu/6.102`

1. `a` , `a`
2. `a` , `b`
3. `ab` , `a`
4. `ab` , `ab`

What does this code do?

- Python and TypeScript behave the same way here
- it helps to draw a diagram

```python
s = "a"
t = s
s += "b"
print(s,t)

s.upper()
print(s,t)

v = []
v.append(s)
w = v
w.append(t)
print(v,w)
```

```typescript
let s = "a";
let t = s;
s += "b";
console.log(s,t);

s.toUpperCase();
console.log(s,t);

let v = [];
v.push(s);
let w = v;
w.push(t);
console.log(v,w);
```

Exercise: yellkey.com/**grow**

Nanoquiz: yellkey.com/**consider**

Clicker: `clicker.mit.edu/6.102`

1. `A` , `a`
2. `AB` , `a`
3. `AB` , `AB`
4. `ab` , `a`

What does this code do?

- Python and TypeScript behave the same way here
- it helps to draw a diagram

```python
s = "a"
t = s
s += "b"
print(s,t)

s.upper()
print(s,t)

v = []
v.append(s)
w = v
w.append(t)
print(v,w)
```

```typescript
let s = "a";
let t = s;
s += "b";
console.log(s,t);

s.toUpperCase();
console.log(s,t);

let v = [];
v.push(s);
let w = v;
w.push(t);
console.log(v,w);
```

Exercise: yellkey.com/**grow**

Nanoquiz: yellkey.com/**consider**

Clicker: `clicker.mit.edu/6.102`

1. `['ab']` , `['a']`
2. `['ab','a']` , `['ab']`
3. `['ab']` , `['ab','a']`
4. `['ab','a']` , `['ab','a']`

Nanoquiz

- This quiz is just for you and your own brain:
  - closed-book, closed-notes
  - nothing else on your screen
- Lower your laptop screen when you're done

yellkey.com/**consider**

Problem Set 0

typical cycle: alpha, code review, beta

ask questions on **Piazza**: expect a conversation

come to **lab hours**: check the course calendar for changes to the lab hour schedule

slack days on pset deadlines must be applied **in advance**

read the **collaboration policy**

Test-first programming

What are the steps of test-first programming?

Partitioning

If you haven't already,
find a partner and
collaborate on the exercise:
yellkey.com/**grow**

Partition the inputs and outputs of this method found in `quadraticRoots.ts`

Write down your partitions in the comment in `describe('quadraticRoots')`

```
/**
 * Solves quadratic equation ax^2 + bx + c = 0.
 *
 * @param a quadratic coefficient, requires a !== 0
 * @param b linear coefficient
 * @param c constant term
 * @returns a list of the real roots of the equation
 */
function quadraticRoots(a: number, b: number, c: number): Array<number>
```

Choosing test cases

Choose test cases for the `intersect` function
so that you cover the partitions in `describe('intersect')`

Write down your test cases, and the subdomains they cover, as `it()` calls

- Just put test cases in comments; don't need to write assertions
- Just cover every subdomain with some test case; don't do full Cartesian product

```
/**
 * Intersects two sets of numbers.
 * For example, intersect({1, 5}, {5, -2}) returns {5}.
 *
 * @param setA another set of numbers
 * @param setB another set of numbers
 * @returns the set { x : x is in both setA and setB }
 */
function intersect(setA: Set<number>, setB: Set<number>): Set<number>
```

Code coverage

```
/**
 * Computes base^exponent mod modulus.
 *
 * @param base     base for exponentiation, >= 0
 * @param exponent exponent for exponentiation, >= 0
 * @param modulus  divisor for modulo operation, > 0
 * @returns base^exponent mod modulus
 */
function powerMod(base: bigint, exponent: bigint, modulus: bigint): bigint {
```

1. Use `npm run coverage` and then view `coverage/index.html`
… What code in `powerMod.ts` is red? Why?

2. Change `powerMod()` to call `powerModFast()` instead and rerun coverage
… Now what code is red? Why?

3. Add test cases to get 100% coverage of `powerModFast()`