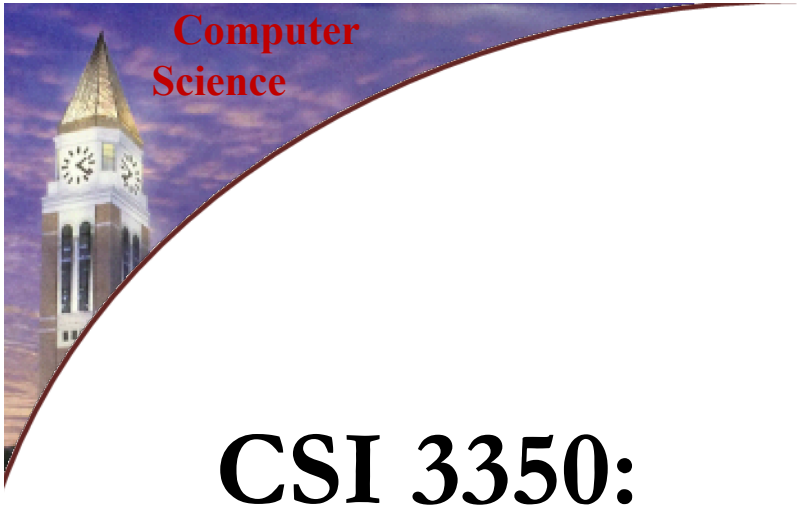


CSI 3350: PROGRAMMING LANGUAGES

Department of Computer Science & Engineering

Oakland University

Road Ahead -

HW05

↓

HW06

↓

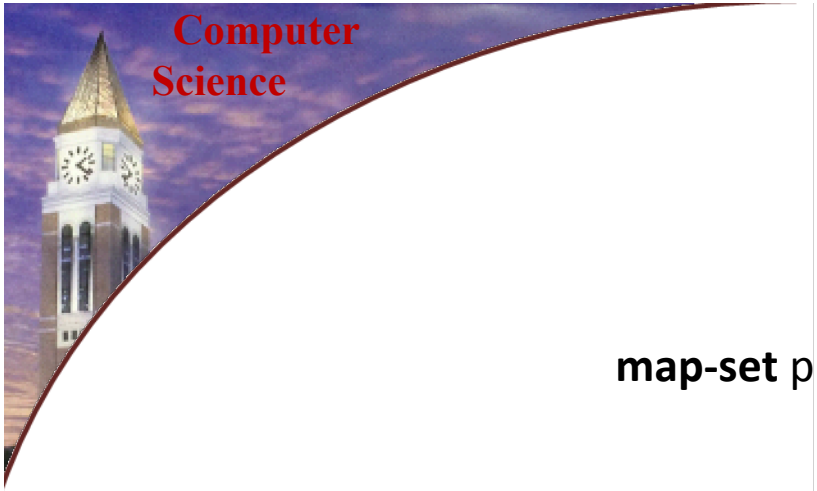
Exam02

↓

HW07

↓

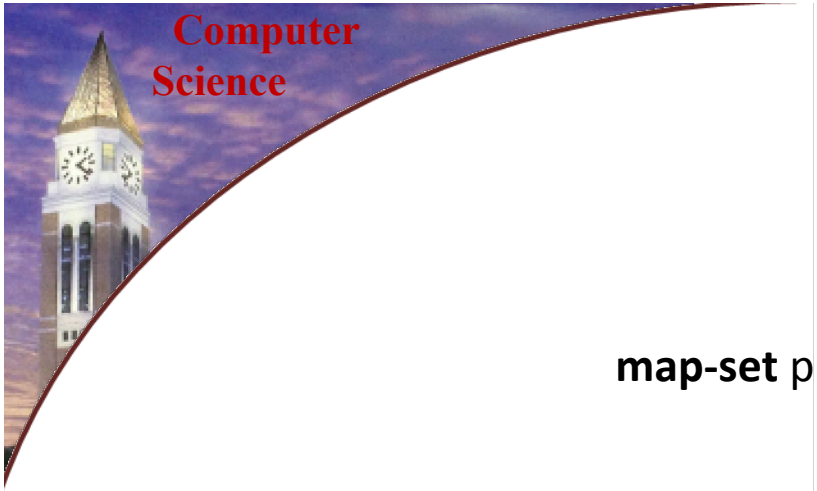
Final Exam : 7pm ~10pm : Dec 09, 2019



HW 4

(Problem 4)

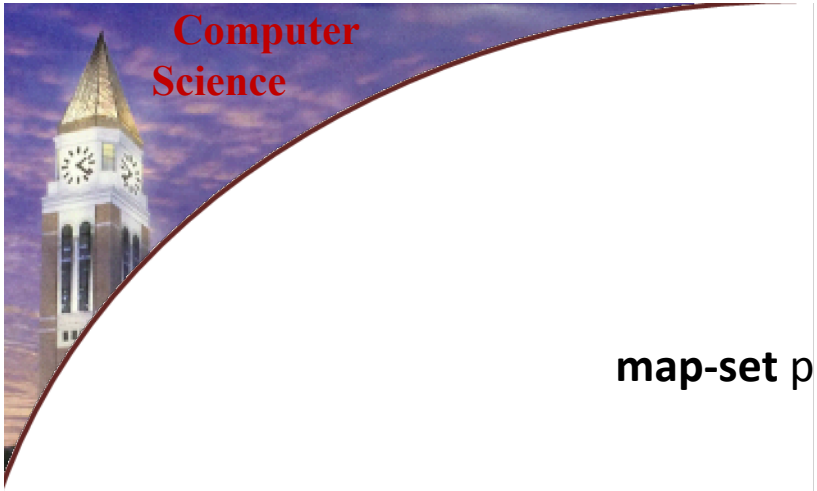
map-set problem



HW 4

(Problem 4)

map-set problem (let us go to our test file)

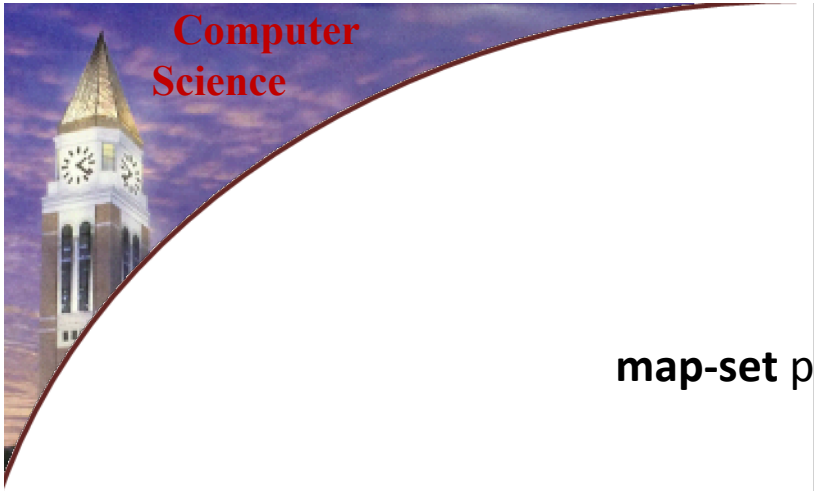


HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)



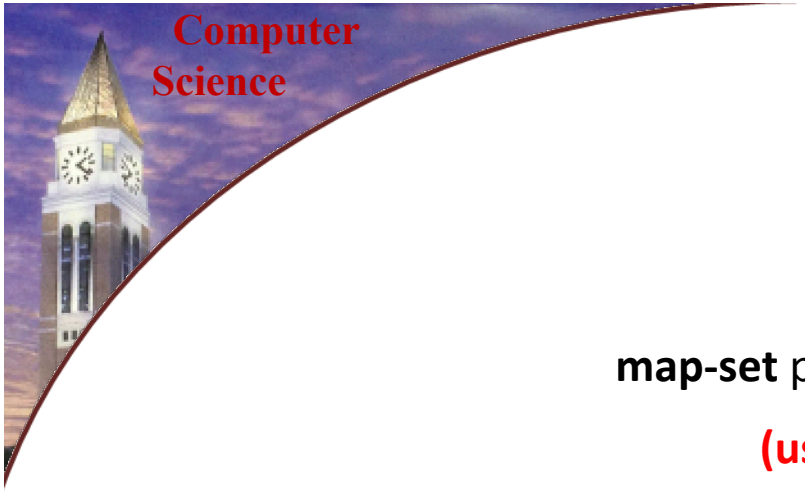
HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

here **{45, 49, 55}** should be represented as a function !



HW 4

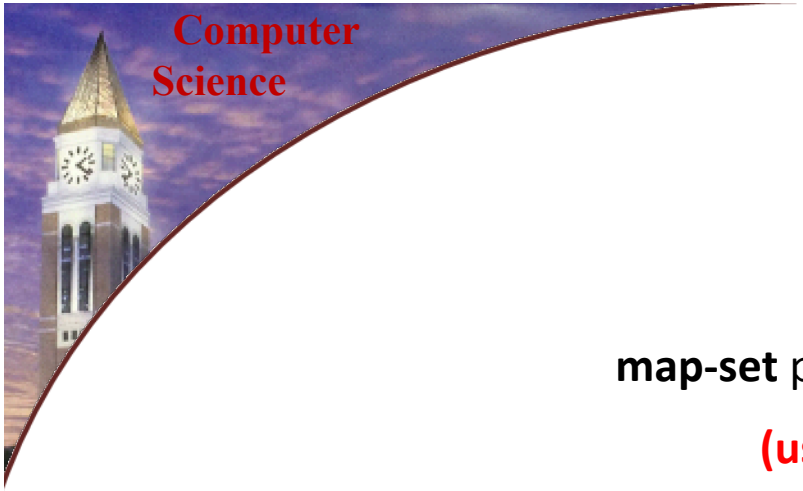
(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

(use the `exist?` function)

here **{45, 49, 55}** should be represented as a function !



HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

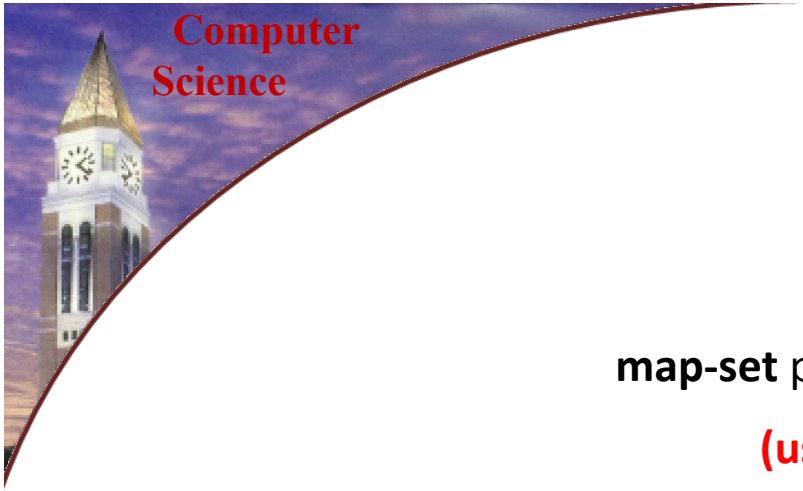
map-set problem (let us go to our test file)

(use the `exist?` function)

here **{45, 49, 55}** should be represented as a function !

```
(define bound 1000)
(define (generate-range) (range (+ bound 1)))

(define (exists? predicate s)
  (define (test? x)
    (if (s x)
        (predicate x)
        #f
        )
    )
  (ormap test? (generate-range))
  )
```



HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

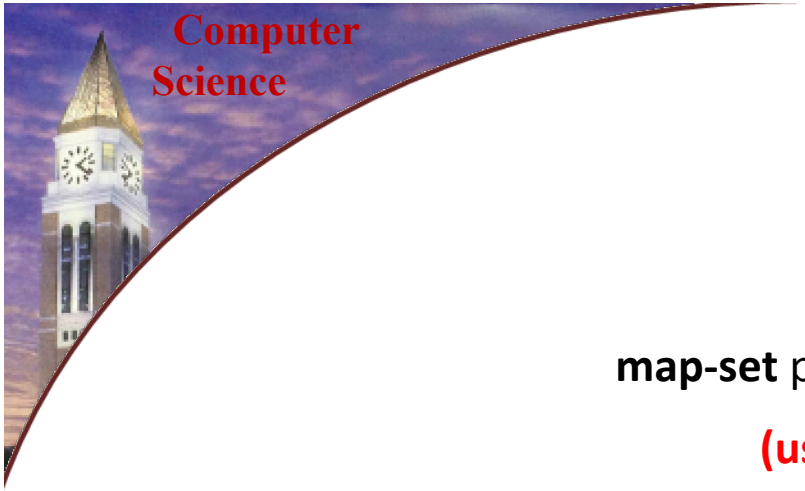
(use the `exist?` function)

here **{45, 49, 55}** should be represented as a function !

to say **45** belongs to the set is the same to saying that there **exists** an number, say **i**, in **{3, 7, 13}** such that **i + 42** is 45

```
(define bound 1000)
(define (generate-range) (range (+ bound 1)))

(define (exists? predicate s)
  (define test? x)
    (if (s x)
        (predicate x)
        #f
        )
    )
  (ormap test? (generate-range))
  )
```



HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

(use the `exist?` function)

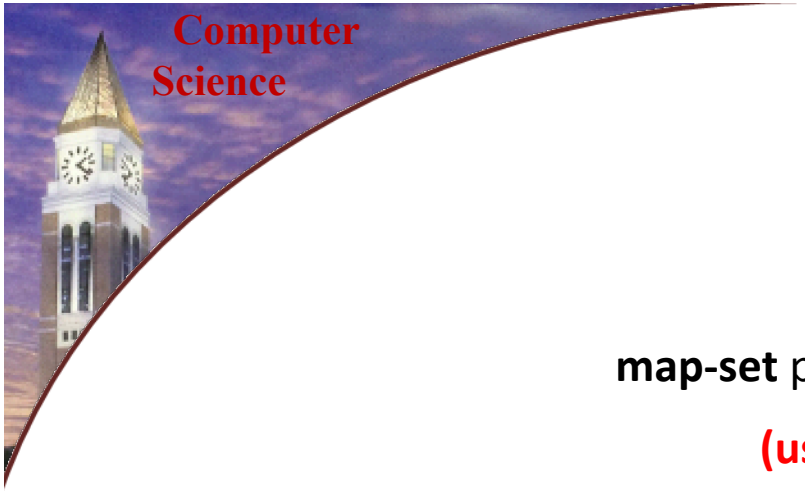
here **{45, 49, 55}** should be represented as a function !

to say **45** belongs to the set is the same to saying that there **exists** an number, say **i**, in **{3, 7, 13}** such that **i + 42** is 45

to translate this idea using the exists? function seen on the right, is the following

```
(define bound 1000)
(define (generate-range) (range (+ bound 1)))

(define (exists? predicate s)
  (define test? x)
    (if (s x)
        (predicate x)
        #f
        )
    )
  (ormap test? (generate-range))
  )
```



HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

(use the `exist?` function)

here **{45, 49, 55}** should be represented as a function !

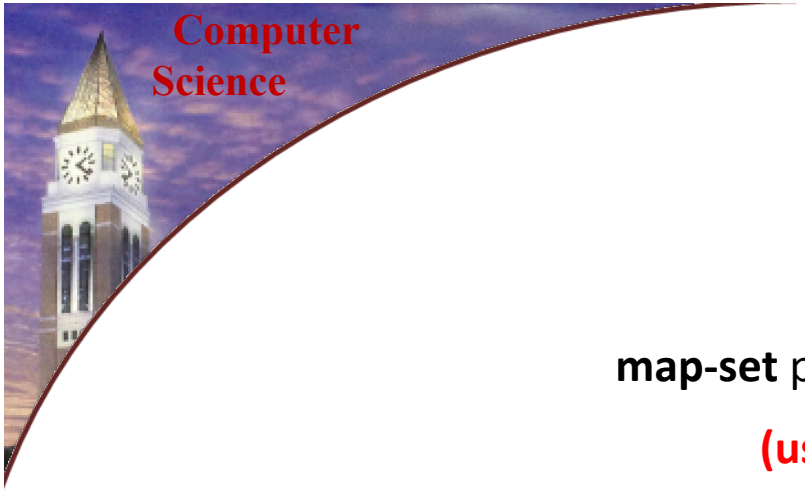
to say **45** belongs to the set is the same to saying that there **exists** an number, say **i**, in **{3, 7, 13}** such that **i + 42** is 45

to translate this idea using the exists? function seen on the right, is the following

```
(define (map-set op s)
  (lambda (x)
    (exists? (lambda (i) (equal? (op i) x))
             s)
    )
  )
```

```
(define bound 1000)
(define (generate-range) (range (+ bound 1)))

(define (exists? predicate s)
  (define test? x)
    (if (s x)
        (predicate x)
        #f
        )
    )
  (ormap test? (generate-range))
  )
```



HW 4

(Problem 4)

Conceptually:
(map-set (lambda(x) (+ x 42)) { 3, 7, 13})
you get the set of
{45, 49, 55}

map-set problem (let us go to our test file)

(use the `exist?` function)

here **{45, 49, 55}** should be represented as a function !

to say **45** belongs to the set is the same to saying that there **exists** an number, say **i**, in **{3, 7, 13}** such that **i + 42** is 45

to translate this idea using the exists? function seen on the right, is the following

```
(define (map-set op s)
  (lambda (x)
    (exists? (lambda (i) (equal? (op i) x))
             s)
    )
  )
```

```
(define bound 1000)
(define (generate-range) (range (+ bound 1)))

(define (exists? predicate s)
  (define test? x)
    (if (s x)
        (predicate x)
        #f
        )
    )
  (ormap test? (generate-range))
  )
```

Road Ahead -

HW05

↓

HW06

↓

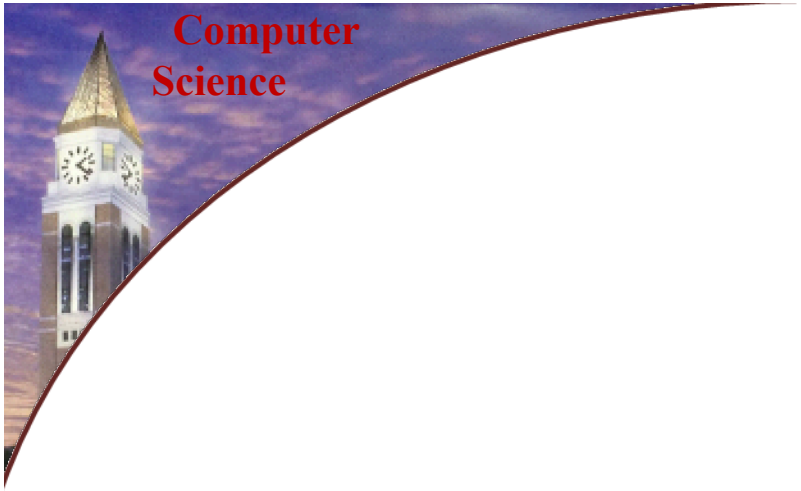
Exam02

↓

HW07

↓

Final Exam : 7pm ~10pm : Dec 09, 2019



Exam 01

(The **`repeat`** loop problem)

HW 4

(Problem 2b)

```
(define-syntax-rule (while condition while-body)
  (letrec (
           (loop (lambda ()
                   (if (not condition)
                       0
                       ( seq while-body (loop) )
                       ))
                 )
           )
    (loop)
    )
  )
```

(1)

Data Type Definition (ADT)

- For complicated types, manual definition tedious
- Plant errors in data type definitions

an observation

- It has recipe!
  - Constructors, Predicates, Extractors

A Tool for Defining Data Types
(less grunt work!)

```
(#%require (lib "eopl.ss" "eopl"))
```

define-datatype

```
(define-datatype type-name predicate-name
  { (variant-name { (field-name predicate)}*)}+
```

define-datatype

```
Env ::= (empty-env )
        | (extend-env var val Env)
```

```
(define-datatype Env Env?
   (empty-env)
   (extend-env (var symbol?)(val number?)(env Env?))
)
```

name of the 2nd variant

name of the 1st field of the 2nd variant

name of the 2nd field of the 2nd variant

name of the 3rd field of the 2nd variant

define-datatype

```
Env ::= (empty-env )
        | (extend-env var val Env)
```

```
(define-datatype Env Env?
    (empty-env)
    (extend-env (var symbol?)(val number?)(Env Env?))
)
```

`var` needs to be of type `number`

`var` needs to be of type `symbol`

`Env` needs to be of type `Env`

What Do We Get?

```
(define-datatype Env Env?
   (empty-env)
   (extend-env (var symbol?)(val number?)(env Env?))
)
```

```
> (Env? #f)
#f
> (Env? (empty-env))
#t
> (Env? (extend-env 'x 20 (empty-env)))
#t
```

What Do We NOT Get?

```
(define-datatype Env Env?
    (empty-env)
    (extend-env (var symbol?)(val number?)(env Env?))
)
```

- We do NOT get
  - Extractors: `Env->var`, `Env->val`, `Env->env`
  - Predicates for variants: `empty-env?`, `extend-env?`

`cases` Syntax Abstraction

`cases` understands `define-datatype`

```
(define-datatype Env Env?
   (empty-env)
   (extend-env (var symbol?)(val number?)(env Env?))
)
```

```
(define apply-env
   (lambda (env search-var)
     (cases Env env)
       (empty-env () (report-no-binding-found search-var))
       (extend-env
          (saved-var saved-val saved-env)
            (if (eqv? search-var saved-var)
                 saved-val
                (apply-env saved-env search-var))))))
```

```
(define-datatype Env Env?
   (empty-env)
   (extend-env (var symbol?)(val number?)(env Env?))
)
```

```
(define (apply-env env search-var)
    (cases Env env
      (empty-env () (raise "No such variable found"))
      (extend-env
         (saved-var saved-val saved-env)
           (if (eqv? search-var saved-var)
               saved-val
               (apply-env saved-env search-var)))))
```