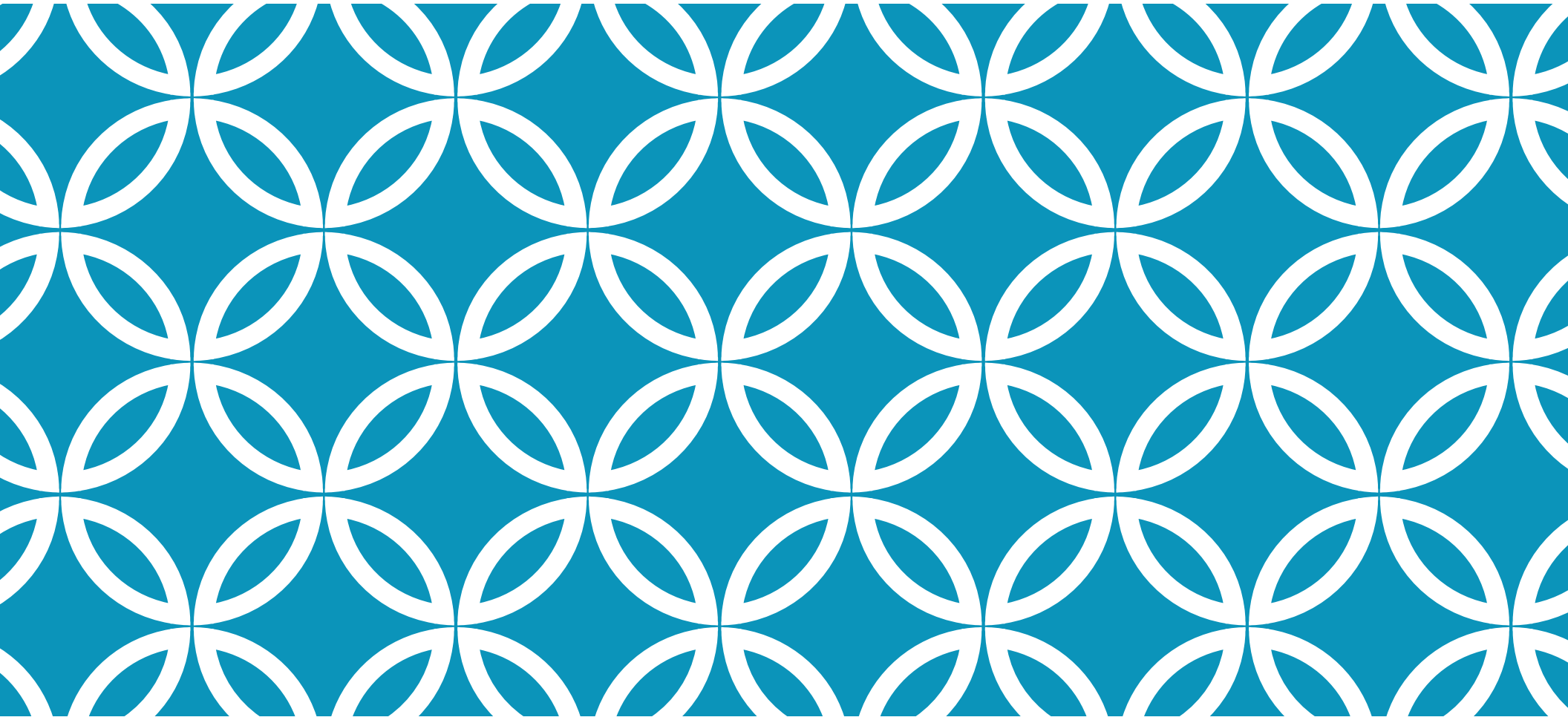

COMP 348 PRINCIPLES OF PROGRAMMING LANGUAGES

Tutorial #5

Functional Programming

LISP (Continued)

VARIABLES AND BINDING

- Binding is a mechanism for implementing lexical scope for variables.
- The **let** syntactic form takes two arguments: a list of bindings and an expression (the body of the binding) in which to use these bindings.

```
( let
      ( (binding1)
        (binding2)
          ...          )
      (expression)  )
```

VARIABLES AND BINDING (EXAMPLES)

- **u** and **v** are let-bound variables; they are only visible within the body of the let.

Examples:

```
> (let ((u 9) (v 2)) (* u v))
18

 > (let ((u 3) (v 5)) (+ (* 2 u) v))
11
```

NESTED BINDING

- A binding can have different values at the same time.

```
(let ((a 1))
   (let ((a 2))
      (let ((a 3))
         ...)))
```

- The inner binding for a variable shadows the outer binding and the region where a variable binding is visible is called its scope.

<u>Example:</u>

> (let ((u 2)(w 1)) (let ((u 4)(v 3)) (let ((u 6)(v (+ v w 4))) (/ u v))))

3/4

NESTED BINDING USING LET*

- Use **let*** when the value of one new variable to depend on the value of another variable established by the same expression.
- A **let*** is functionally equivalent to a series of nested lets.

Example:

```
> (let* ((u 3) (v (* 5 u)))(/ v u))
5
```

DEFINING FUNCTIONS

- We can define new functions using **defun**. Following is the syntax of a defun function.

```
( defun  name ( formal parameter list )
         body )
```

Examples:

1/ Consider function **absdiff** takes two arguments and returns their absolute difference:

```
> (defun absdiff (p1 p2) (if (> p1 p2) (- p1 p2) (- p2 p1)))
ABSDIFF
> (absdiff 9 18)
9
```

DEFINING FUNCTIONS (CONT..)

2/ Construct a function, called **evenf**, which tests whether its argument is an even number using the reminder function **REM** to get the rest of the division.

```
> (defun evenf(num) (if (zerop (rem num 2)) num nil))

EVENF

> (evenf 9)

NIL

> (evenf 80)

80
```

DEFINING FUNCTIONS (CONT..)

3/ Construct a function, called **oddf**, which tests whether its argument is an odd number.

```
 > (defun oddf(num) (if (= (rem num 2) 1) num nil))

ODDF

> (oddf 17)

17

> (oddf 70)

NIL
```

ANONYMOUS FUNCTIONS

- An anonymous function is one that is defined, and possibly called, without being bound to an identifier.
- Unlike functions defined with **defun**, anonymous functions are not stored in memory. The general syntax of an anonymous function in Lisp (also called **lambda** expression) is

(lambda (*formal parameter list*) (*body*))

Examples:

1/ An anonymous function can be applied in the same way that a named function can, e.g.,

```
> ((lambda (w) (+ w (* w w))) 2)
6
```

ANONYMOUS FUNCTIONS (CONT..)

2/ Consider a function that takes a list as an argument and returns a new list whose elements are the elements of the initial list multiplied by 5.

We can perform the multiplication with an anonymous function, and deploy **mapcar** to apply the anonymous function to the elements of the list as follows:

mapcar takes as its arguments a function and one or more lists and applies the function to the elements of the list(s) in order.

> (mapcar (lambda (w) (* w 5)) '(1 0 3 7 2 11))

(5 0 15 35 10 55)

FUNCTION EQL VS EQUAL

- Variables are essentially pointers.
- Function **eql** will return true if its arguments point to the same object, whereas function **equal** returns true if its arguments have the same value.

Examples:

```
> (setf u '(a b c))
(A B C)

> (setf v '(a b c))
(A B C)

> (eql u v)
NIL
```

FUNCTION EQL VS EQUAL (CONT..)

```
> (equal u v)
T

> (setf w u)
(A B C)

> (eql u w)
T

> (equal u w)
T
```

EXERCISE#1

Determine the number of elements of the following lists.

1) (A (B C ()) D ())

2) (1 a 3 b 5 1 (c d) a (c c))

3) ((B) (A) (A B) (a b (A B)) 7)

4) (comp348 (fall) (2018) (principles of (Prog Lang)))

EXERCISE#2

Provide the results of calls of the following functions:

Lists	**Results**
(cons '(a b) '(c d))	
(cons () '(a b c ()))	
(cons 'a (cons '(b 2)(cons 'c '(d))))	
(list '(* (5 + 2)(+ 3 2)))	
(list '(1) (cons '(a 2) '(c)) (cons '(b 3 ()) ()))	
(list (/ 15 (float 2)) () (cons '(c 3)()))	
(append (list 'c ()) '(a 2))	
(append '() '(b d))	

EXERCISE#3

Provide the results of calls of the following functions:

Lists	**Results**
(cons (car '(u v)) (cdr '(w x y z)))	
(cdr (cons 'u '(v w x)))	
(car (car (cdr (cdr '(u v (w x) y)))))	
(cons 'cdr (cons '(cdr '(u v w)) ()))	
(cdr (append (cdr '(a b)) '(d e)))	
(cons (cdr (append (cdr '(a b)) '(d e))))	

EXERCISE#4

A recursive function to find the power of a number.

Example: (power 2 5) => 32

EXERCISE#5

Find the factorial of a given number. Consider the following definition.

N! = 1 if N = 1

N! = N * (N – 1)! if N > 1

Example: (factorial 5) => 120

EXERCISE#6

Find the Fibonacci number of a given N. Consider the following definition.

Fib(N) = 1 for N = 0 or N =1

Fib(N) = Fib(N-1) + Fib(N-2) for N > 1

Example: (fibonacci 6) => 13