Paul Burgwardt

NOTES [8/26 – 9/18]

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| 3 | 3 | Mode for expressions is prefix, where operator first then arguments |
| (+ 2 3 5) | 10 |
| (+ 2 (\* 3 5)) | 17 |
| (\* 2 (+ 3 5)) | 16 |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define x 3)  (+ 2 (\* x 5)) | 17 | Creates binding between a name and expression |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define square  (lambda (x)  (\* x x)))  (square 10) | 100 | Defined function named square that returns the square of the input |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| ((lambda (y) (\* y y)) 10) | 100 | Anonymous function |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define compose  (lambda (f g)  (lambda (x)  (f (g x)))))  (define fourth-power (compose square square))  (fourth-power 2) | 16 | Takes two functions as input and outputs another function (described as lambda expression) |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define boo(cons ‘apple’ ‘()))  boo | (apple quote ()) | Constructs a list with element ‘apple’ |
| (car ‘(a b c)) | a | Gets first element |
| (cdr ‘(a b c)) | (b c) | Gets everything except first element |
| (cons ‘a ‘(b c)) | (a b c) | Combines the elements from each array |
| (cons (car ‘(a b c)) (cdr ‘(d e f))) | (a e f) | Combines first element from (a b c) and everything except the first element from (d e f) |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define reciprocal  (lambda (n)  (if (= n 0)  “Oops! Not valid!”  (/ 1 n))))  (reciprocal 2) | 1/2 |  |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (quote (1 2 3 4 5)) | (1 2 3 4 5) | Creates a list |
| (quote (“this” “is” “a” “list”)) | (“this” “is” “a” “list”) | Gets first element |
| (quote (+3 4)) | (+ 3 4) | Treat as list, not operation |
| ‘(1 2 3 4) | (1 2 3 4) | Short-hand notation for quote |
| (car (cdr ‘(a b c))) | b |  |
| (cdr (cdr ‘(a b c))) | (c) | Maintains list format |
| (car ‘((a b) (c d))) | (a b) |  |
| (cdr ‘((a b) (c d))) | ((c d)) |  |

NOTE; The cdr of a list with one element is (), the empty list.

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define length  (lambda (ls)  (if (null? ls )  0  (+ (length (cdr ls)) 1)))) |  | Recursively defines a length function |
| (length ‘()) | 0 | Satisfies base-case, so zero |
| (length ‘(a)) | 1 | One element |
| (length ‘(a b)) | 2 | Two elements |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define list-copy  (lambda (ls)  (if (null? ls)  ‘()  (cons (car ls)  (list-copy (cdr ls)))))) |  | Recursively defines a list-copy function |

|  |  |  |
| --- | --- | --- |
| **Expression** | **Result** | **Notes** |
| (define list-from-to  (lambda (low high)  (if (> low high)  ‘()  (cons low (list-from-to(+ low 1) high))))) |  | Recursive function that creates list from low to high |
| (list-from-to 1 10) | (1 2 3 … 10) |  |
| (define my-append  (lambda (ls1 ls2)  (if (null? ls1)  ls2  (cons (car ls1) (my-append (cdr ls1) ls2))))) |  | Recursively appends two lists |
| (my-append ‘(1 2 3) ‘(4 5 6)) | (1 2 3 4 5 6) |  |
| (define my-reverse  (lambda (ls)  (cond ((null? ls) ‘())  (else (my-append (my-reverse (cdr ls))  (list (car ls))))))) |  | Recursively reverses a list |
| (my-reverse ‘(1 2 3 4)) | (4 3 2 1) |  |