For each expression or set of expressions, give the value and type of the value returned by evaluating the last expression in the set.

<table>
<thead>
<tr>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ((lambda (x) (+ x y)) 7)</td>
<td>error, undefined variable y</td>
</tr>
<tr>
<td>2. ((lambda (x) (let ((y 4)) (+ x y))) 7)</td>
<td>11</td>
</tr>
<tr>
<td>3. (lambda (x) (x 4 5))</td>
<td>compound procedure</td>
</tr>
<tr>
<td>4. (lambda (a b c) (+ a b))</td>
<td>compound procedure</td>
</tr>
<tr>
<td>5. (lambda (x y) (lambda (x) (y x)))</td>
<td>compound procedure</td>
</tr>
<tr>
<td>6. ( ((lambda (x y) (lambda (z) (x y z)) + 2) 4) 6)</td>
<td>number</td>
</tr>
</tbody>
</table>

Using the substitution model, first substitute + and 2 for x and y, and call the outer level lambda: ((lambda (z) (+ 2 z)) 4); then call the lambda with 4 bound to z to get (+ 2 4). Note that without the 4 and the outer parens, the value is a compound procedure: (lambda (z) (+ 2 z)), with type num->num.
7.\((\text{lambda } (x))\)  
\((\text{let } ((a\ 1)\ (b\ 5))\)  
\((\text{if } x\ a\ b))\)  
\((>\ 20\ 10))\)  
\(1\)  
\(\text{number}\)  
The lambda is called with the value of \((>\ 20\ 10)\), with is #t; so the value of a is returned.

8.\(\text{(define } x\ \text{+})\)  
\((\text{let } ((a\ 3))\)  
\((\text{list } x\ a\ a))\)  
\((+\ 3\ 3)\)  
\(\text{pair(num->num, list(num))}\)  
The return value is not evaluated; the list is just constructed. The type is a pair rather than a list because the list type is specified with one type for its members. So the type of \((+\ 3\ 3)\) also would be list(A), but the above pair type is more specific, so preferred.

9.\(\text{(define } (\text{foo } a\ b)\)\)  
\((\text{let } ((x\ 6)\ (c\ (+\ a\ 5)))\)  
\((+\ b\ x\ c))\)  
\(\text{number}\)  
The lambda is called with arguments that bind x to 1, y to 2, f to the procedure foo. The expression \((\text{foo } 1\ 2)\) is then evaluated to get 14.

Extra problem (not to worry about now):
10.\(\text{(let } ((a\ 10)\ (b\ 2))\)  
\((\text{let } ((c\ (+\ a\ b)))\)  
\((*\ a\ c))\)  
\(120\)  
\(\text{number}\)  
Note: The second let is needed because the value of a variable is not bound until the entire list of variable-value pairs is evaluated. In this example, the value of a or b can't be used in defining c in the first let's list of variables.