Function Compositions

Class 8
2.2 Composition of Functions

- this section introduces the concept of function composition
- let $g : A \to B$ be a function
- let $f : B \to C$ be a different function
- then $f \circ g$ is a composition, a new function
- $f \circ g : A \to C$

\[(f \circ g)(x) = f(g(x))\]

- composition is associative but not commutative
  \[(\text{\LaTeX}: (f \circ g)(x) = f(g(x)))\]
Example

- let $h(x) = 2x + 3$ and $g(x) = 3x + 2$
- find $(h \circ g)(x)$

\[
(h \circ g)(x) = h(g(x)) \\
= h(3x + 2) \\
= 2(3x + 2) + 3 \\
= 6x + 4 + 3 \\
= 6x + 7
\]
Connection to Programming

• everything we do in this course has a direct bearing on how programs work

• discrete math is the basis of computer science, just like calculus is the basis of physics

• suppose we have a binary tree and we wish to know the minimum possible depth of the tree, given the number of vertices

• as shown in the table on page 107, there is a function that maps the number of vertices to the minimum depth
• the minimum depth function is a combination, or composition, of the floor function and the base-2 log function
• since floor and log are built into every programming language, we don’t have to write our own depth function from scratch
• we can simply compose the desired function from existing pieces
Minimum Depth

• mathematically:

\[ \text{min\_depth}(n) = (\text{floor} \circ \lg)(n) = \text{floor}(\lg(n)) \]

• in C++:

```cpp
unsigned min_depth(unsigned vertices)
{
    return static_cast<unsigned>(floor(log2(vertices)));
}
```
List Functions

• four useful functions when dealing with lists are
  
  seq  return a sequence from 0 to the argument
  dist distribute the first, singleton argument across the second, list argument, to generate a new list of tuples
  pairs return a list of pairs of corresponding elements of the two list arguments
  map  similar concept to dist, but applies a function to each list element

• all four are built into many modern languages
**seq: Generate a Sequence**

- **Python:** `for i in range(4):` (gives 0, 1, 2, 3)
- **BASH:** `$ seq 4` (gives 1, 2, 3, 4)
- **this course:** `seq(4) = ⟨0, 1, 2, 3, 4⟩`

- it’s very easy to be confused by whether the list starts at 0 or 1
- and whether the end is inclusive or exclusive
dist: Distribute an Element Over a List

• in Python, this is list comprehension
• [('a', x) for x in [1, 2, 3]] gives [(a, 1), (a, 2), (a, 3)]
• BASH: $ echo a{1,2,3} gives a1 a2 a3
• this course: dist(a, ⟨1, 2, 3⟩) = ⟨(a, 1), (a, 2), (a, 3)⟩
pairs: Create Pairs From Two Lists

- Python zip: `list(zip([1, 2, 3], ['a', 'b', 'c']))` gives `[(1, 'a'), (2, 'b'), (3, 'c')]`
- BASH: `$ paste numbers letters` (assuming appropriate file contents)
- this course: `pairs(⟨1, 2, 3⟩, ⟨a, b, c⟩) = ⟨(1, a), (2, b), (3, c)⟩`

- the lists **must** be the same length, otherwise `pairs` is undefined
- in Python extra elements in one list are ignored
- in BASH extra elements are paired with the empty string
map: Apply a Function to a List

- Python: `list(map(lambda x : x**2, [0, 1, 2, 3]))` gives `[0, 1, 4, 9]`
- this course: let $L = \langle 0, 1, 2, 3 \rangle$ and $f(n) = n^2$ then `map(f, L)` gives us $\langle 0, 1, 4, 8 \rangle$
2.3 Function Characteristics

- There are two characteristics that some functions have that are extremely important:
- A function is **injective**, or 1-to-1, if no two elements of the domain map to the same element of the range.

```
  domain       codomain
         a     c
         b     c
         d     e
```

Surjection

• a function is **surjective**, or “onto”, if the codomain and range are the same set

<table>
<thead>
<tr>
<th>domain</th>
<th>codomain</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>b</td>
<td>f</td>
</tr>
<tr>
<td>c</td>
<td>[x] h</td>
</tr>
</tbody>
</table>

In the diagram, element c is not mapped to any element in the codomain, and h is not an element of the domain.
Bijection

• some functions are injections but not surjections (give an example)
• some are surjections but not injections (give an example)
• some functions are neither injections nor surjections (give an example)

• but some functions are both injections and surjections
• these are called bijections, or “1-to-1 and onto”
• no two arrows point to the same element of the codomain
• every element of the codomain is pointed to by some arrow
Inverses

• bijections are so important because they are unique in being invertible

• a bijective function has an inverse function (which is also, by definition, a bijection)

• if $f$ is a function, we denote its inverse by $f^{-1}$
  (LaTeX: $f^{-1}$)

• for example, let $f$ be the function $f(n) = n + 1$ which maps each odd integer to an even integer

• then $f^{-1}(n) = n - 1$ is the inverse of $f$ which maps each even integer to an odd integer (in the same pairing as the original)

• note that $f^{-1}(f(n)) = n$ and $f(f^{-1}(n)) = n$ for any bijection $f$
The Pigeonhole Principle

at the top of page 118 is a topic that is incredibly important even though it seems so obvious

The Pigeonhole Principle

If $n$ items are put uniquely into $m$ containers, and if $n > m$, then at least one container will have at least two items.

your text lists some examples at the bottom of page 118, but several of these are worded so sloppily as to be incorrect

your text incorrectly states “if a six-sided die is tossed seven times, one side will come up twice”

the correct statement is: “if a six-sided die is tossed seven times, at least one side will come up at least twice”