recursion template

Tuesday, October 17, 2023 5:39 PM

```
fun list-sum(num-lst :: List<Number>) -> Number:
  cases(List) num-lst:
    | empty => 0
    | link(fst, rst) => fst + list-sum(rst)
  end
end
```

```
fun double(num-lst :: List<Number>) -> List<Number>:
  cases(List) num-lst:
    | empty => empty
    | link(fst, rst) => link(2 * fst, double(rst))
  end
end
```

```
fun my-pos-nums(l :: List<Number>) -> List<Number>:
  cases (List) l:
    | empty => empty
    | link(f, r) =>
      if f > 0:
        link(f, my-pos-nums(r))
      else:
        my-pos-nums(r)
      end
  end
end
```

```
fun len(lst :: List<String>) -> Number:
  cases(List) lst:
    | empty => 0
    | link(f, r) => 1 + len(r)
  end
end
```

```
fun stack-boxes(colors :: List<String>) -> Image:
  cases(List) colors:
    | empty => rectangle(0, 0, "solid", "white")
    | link(f, r) =>
      above(
        rectangle(30, 30, "solid", f),
        stack-boxes(r))
  end
end
```

```
fun contains-cat(str-lst :: List<String>) -> Boolean:
  cases(List) str-lst:
    | empty => false
    | link(fst, rst) =>
      if fst == "cat":
        true
      else:
        contains-cat(rst)
      end
  end
end
```

A recursive function on lists will typically contain:

- A cases expression that asks if the list is empty or a link of a first element to a rest
- Some simple answer for the empty case
- Some computation on the first element
- A recursive call on the rest of the list
- A way to connect the computation on the first element to the recursive call on the rest of the list