Pseudocode: Lists, Dictionaries and Side-Effects

Dealing with side-effects

- Comparing two lists for duplicate items
  - Nested loop

```
Procedure FindOverlap(l1,l2)
  overlap = []
  foreach x in l1 {
    foreach y in l2 {
      if (x == y) {
        overlap = overlap ++ [x]
      }
    }
  }
  return(overlap)
End FindOverlap
```

Dealing with side-effects

- Comparing two lists for duplicate items
  - Nested loop
- What if the lists are sorted?
  - Need not start inner iteration from the beginning
  - Use `first()` and `rest()` to cut down the list to be scanned

```
Procedure FindOverlap2(l1,l2)
  overlap = []
  if (length(l2) == 0) {
    return(overlap)
  }
  else {
    y = first(l2)
    l2 = rest(l2)
  }
  foreach x in l1 {
    while (y < x and length(l2) > 0) {
      y = first(l2)
      l2 = rest(l2)
    }
    if (x == y) {
      overlap = overlap ++ [x]
    }
  }
  return(overlap)
End FindOverlap2
```

Dealing with side-effects

- Comparing two lists for duplicate items
  - Nested loop
- What if the lists are sorted?
  - Need not start inner iteration from the beginning
  - Use `first()` and `rest()` to cut down the list to be scanned
- Second list has been modified inside the procedure
  - Side-effect!

```
Procedure FindOverlap2(l1,l2)
  overlap = []
  if (length(l2) == 0) {
    return(overlap)
  }
  else {
    y = first(l2)
    l2 = rest(l2)
  }
  foreach x in l1 {
    while (y < x and length(l2) > 0) {
      y = first(l2)
      l2 = rest(l2)
    }
    if (x == y) {
      overlap = overlap ++ [x]
    }
  }
  return(overlap)
End FindOverlap2
```

Dealing with side-effects

- Comparing two lists for duplicate items
  - Nested loop
- What if the lists are sorted?
  - Need not start inner iteration from the beginning
  - Use `first()` and `rest()` to cut down the list to be scanned
- Second list has been modified inside the procedure
  - Side-effect!
- Instead, make a copy of the input parameter

```
Procedure FindOverlap3(l1,l2)
  overlap = []
  if (length(l2) == 0) {
    return(overlap)
  }
  else {
    myl2 = l2
    y = first(myl2)
    myl2 = rest(myl2)
  }
  foreach x in l1 {
    while (y < x and length(myl2) > 0) {
      y = first(myl2)
      myl2 = rest(myl2)
    }
    if (x == y) {
      overlap = overlap ++ [x]
    }
  }
  return(overlap)
End FindOverlap3
```

Dealing with side-effects

- Comparing two lists for duplicate items
  - Nested loop
- What if the lists are sorted?
  - Need not start inner iteration from the beginning
  - Use `first()` and `rest()` to cut down the list to be scanned
- Second list has been modified inside the procedure
  - Side-effect!
- Instead, make a copy of the input parameter

```
Procedure FindOverlap3(l1,l2)
  overlap = []
  if (length(l2) == 0) {
    return(overlap)
  }
  else {
    myl2 = l2
    y = first(myl2)
    myl2 = rest(myl2)
  }
  foreach x in l1 {
    while (y < x and length(myl2) > 0) {
      y = first(myl2)
      myl2 = rest(myl2)
    }
    if (x == y) {
      overlap = overlap ++ [x]
    }
  }
  return(overlap)
End FindOverlap3
```

Deleting a key from a dictionary

- Delete a key from a dictionary?
  - Copy all keys and values except the one to be deleted to a new dictionary
  - Copy back the updated dictionary

```
Procedure DeleteKey(d,k)
  myd = {}
  foreach key in keys(d) {
    if (k ≠ key) {
      myd[key] = d[key]
    }
  }
  d = myd
End DeleteKey
```

Deleting a key from a dictionary

- Delete a key from a dictionary?
  - Copy all keys and values except the one to be deleted to a new dictionary
  - Copy back the updated dictionary
- In this case, the side effect in the procedure is intended
  - Use side-effects to update a collection inside a procedure
  - Sorting a list in place

```
Procedure DeleteKey(d,k)
  myd = {}
  foreach key in keys(d) {
    if (k ≠ key) {
      myd[key] = d[key]
    }
  }
  d = myd
End DeleteKey
```

Deleting a key from a dictionary

- Delete a key from a dictionary?
  - Copy all keys and values except the one to be deleted to a new dictionary
  - Copy back the updated dictionary
- In this case, the side effect in the procedure is intended
  - Use side-effects to update a collection inside a procedure
  - Sorting a list in place
- We can also program this without side-effects
  - Return the updated dictionary

```
Procedure DeleteKey2(d,k)
  myd = {}
  foreach key in keys(d) {
    if (k ≠ key) {
      myd[key] = d[key]
    }
  }
  return(myd)
End DeleteKey2
```

Deleting a key from a dictionary

- Delete a key from a dictionary?
  - Copy all keys and values except the one to be deleted to a new dictionary
  - Copy back the updated dictionary
- In this case, the side effect in the procedure is intended
  - Use side-effects to update a collection inside a procedure
  - Sorting a list in place
- We can also program this without side-effects
  - Return the updated dictionary
  - Reassign it after the procedure call

```
Procedure DeleteKey2(d,k)
  myd = {}
  foreach key in keys(d) {
    if (k ≠ key) {
      myd[key] = d[key]
    }
  }
  return(myd)
End DeleteKey2

myd = DeleteKey2(myd,k)
```

Summary

- Be careful of side-effects when working with collections
  - Make a local copy of the argument
- Sometimes, side effects are convenient for updating collections in place
  - Deleting a key in a dictionary
  - Sorting a list
- Can also return a new collection and reassign after the procedure call
  - `myd = DeleteKey2(myd,k)`
  - `mylist = InsertionSort(mylist)`