COMP1002

Computational Thinking and Problem Solving

Lecture 4
Computation II

Lecture 4

› Conditionals
  – List of conditions
  – Chained conditions
  – Nested/multi-way conditions
  – Efficiency Consideration

› Complex Conditionals
  – Decision Trees

› Sorting Numbers

Conditionals

- if cond then A
  - Check the condition cond.
  - If it is true, then execute A.
  - Otherwise, do nothing.

```
if a < 0 then
    a = -a
```

  - Python:

```python
if a < 0:
    a = -a
```

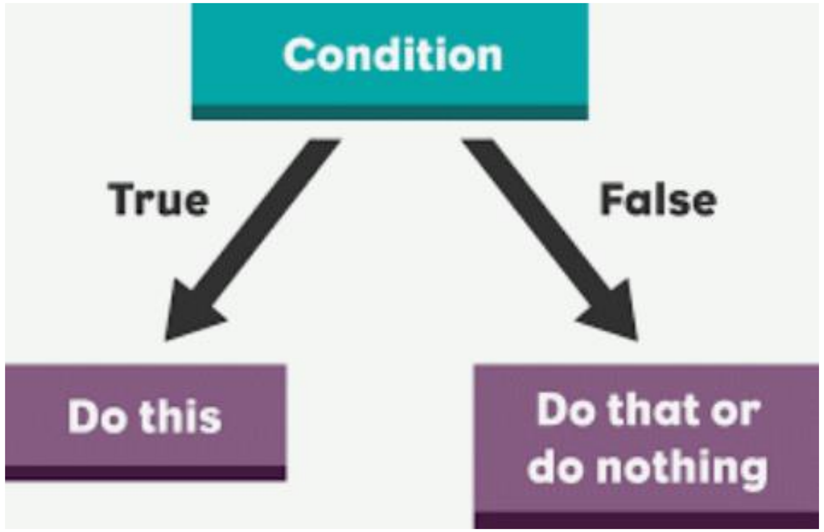
Conditionals

- if cond then A else B
  - Check the condition cond.
  - If it is true, then execute A.
  - Otherwise, execute B.

  - Python:

```python
if a < b:
    answer = a
else:
    answer = b
```

```
if a < b then
    answer = a
else
    answer = b
```

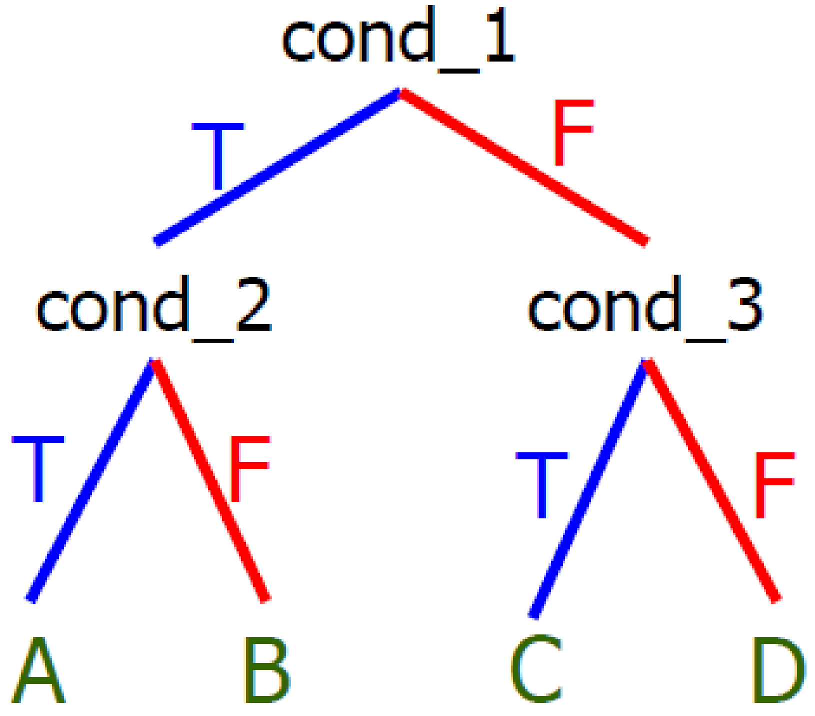


Conditionals

› Conditionals can be nested

```
if cond_1 then
    if cond_2 then A
    else B
else
    if cond_3 then C
    else D
```

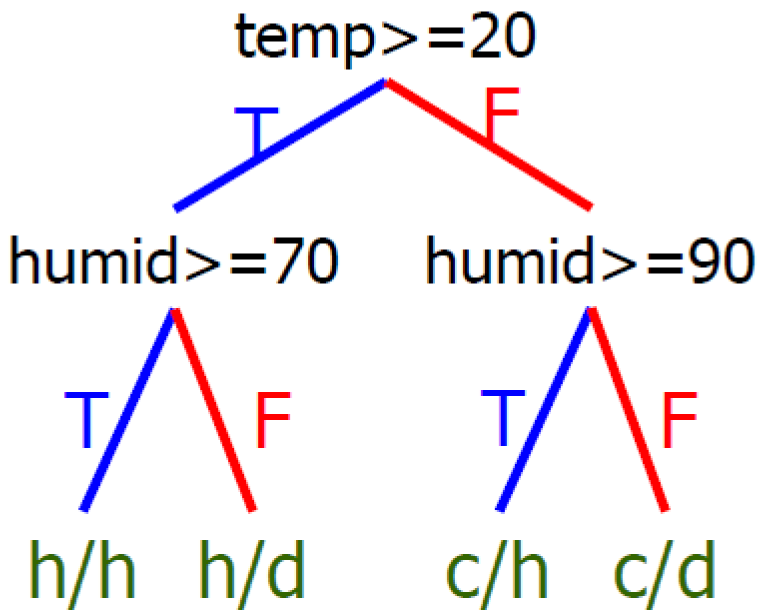
› Four types of situations
  - cond_1 true and cond_2 true => A
  - cond_1 true and cond_2 false => B
  - cond_1 false and cond_3 true => C
  - cond_1 false and cond_3 false => D



Conditionals

› Example

```
if temperature >= 20 then
   if humidity >= 70 then
      print “hot and humid”
   else
      print “hot and dry”
else # temperature < 20
   if humidity >= 90 then
      print “cold and humid”
   else
      print “cold and dry”
```



Conditionals

› Given three unequal numbers, a, b, c, how can the correct ordering be printed?

```
if a > b then
   if b > c then
      print “a > b > c”
   else
      print “a > c > b”
else # this means a < b
   if b < c then
      print “a < b < c”
   else
      print “a < c < b”
```

Is this correct?

Conditionals

- Indentation does matter!
- Consider ordering a dinner dish at a typical Hong Kong restaurant:
  - Cost of dinner main dish is $80
  - Order dinner set with $25 more (salad + drink)
  - Special drink in dinner set can be ordered for $10 more (e.g., fruit punch or lemon coke)
  - All special drinks are cold drinks
  - Standard drink means coffee or tea
  - Standard cold drink at $5 more

Program 1

```
set cost to 80
if dinner set then
   cost = cost + 25
   if special drink then
      cost = cost + 10
   else
      if cold drink then
         cost = cost + 5
return cost
```

Program 2

```
set cost to 80
if dinner set then
   cost = cost + 25
   if special drink then
      cost = cost + 10
else
   if cold drink then
      cost = cost + 5
return cost
```

Which reflect(s) the intention of the restaurant?

Conditionals

Is this correct?

- List of conditions
- Count the number of times for each face of a die

```
reset count1 to count6 value to zero
for i in [1…1000] do
    face = throw a die
    if face = 1 then count1 = count1 + 1
    if face = 2 then count2 = count2 + 1
    if face = 3 then count3 = count3 + 1
    if face = 4 then count4 = count4 + 1
    if face = 5 then count5 = count5 + 1
    if face = 6 then count6 = count6 + 1
```

Conditionals

Is this correct?

› List of conditions
  – Count the number of grades in a class

```
reset countA to countF value to zero
for each student s in class list L do
    g = score of student s
    if g >= 85 then countA = countA + 1
    if g >= 70 then countB = countB + 1
    if g >= 55 then countC = countC + 1
    if g >= 40 then countD = countD + 1
    if g < 40 then countF = countF + 1
```

Conditionals

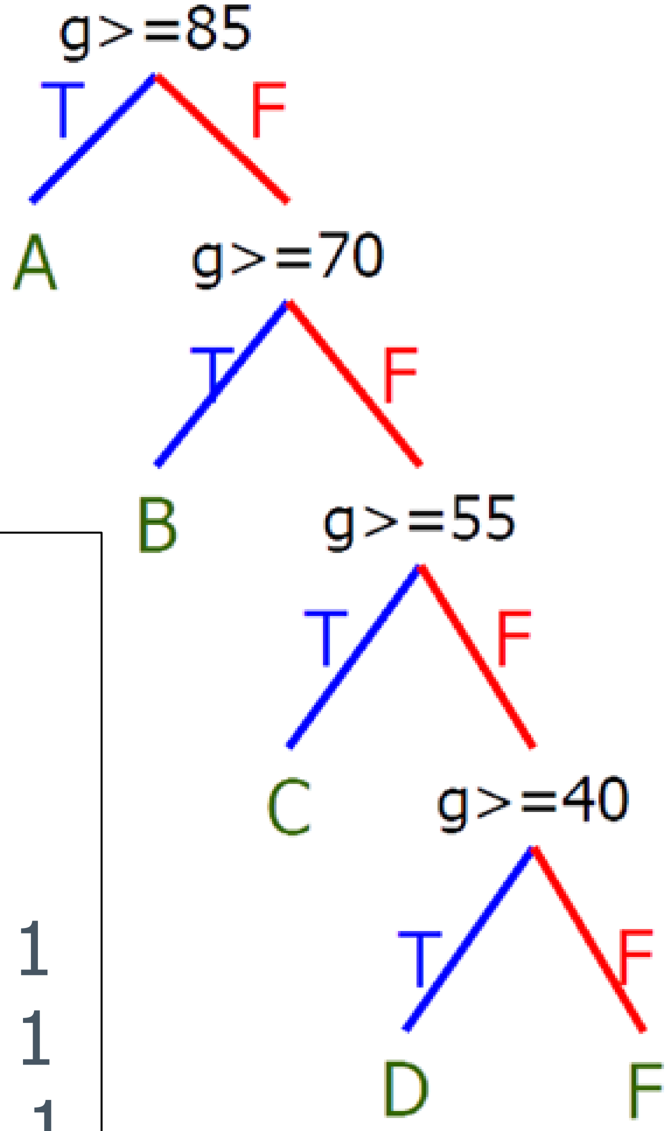
- List of conditions
  - Count the number of grades in a class

```
reset countA to countF value to zero
for each student s in class list L do
    g = score of student s
    if g >= 85 then countA = countA + 1
    if 70 <= g < 85 then countB = countB + 1
    if 55 <= g < 70 then countC = countC + 1
    if 40 <= g < 55 then countD = countD + 1
    if g < 40 then countF = countF + 1
```

Conditionals

› Chained conditions
  – Count the number of grades in a class

```
reset countA to countF value to zero
for each student s in class list L do
    g = score of student s
    if g >= 85 then countA = countA + 1
    else if g >= 70 then countB = countB + 1
    else if g >= 55 then countC = countC + 1
    else if g >= 40 then countD = countD + 1
    else if g < 40 then countF = countF + 1
```



Conditionals

- Exercise
  - Convert the above pseudocode to Python

Conditionals

- Which is easier?
- Which is better?

if g >= 85 then countA = countA + 1 if 70 <= g < 85 then countB = countB + 1 if 55 <= g < 70 then countC = countC + 1 if 40 <= g < 55 then countD = countD + 1 if g < 40 then countF = countF + 1	if g >= 85 then countA = countA + 1 else if g >= 70 then countB = countB + 1 else if g >= 55 then countC = countC + 1 else if g >= 40 then countD = countD + 1 else countF = countF + 1

Are there other approaches?

Conditionals

Can you draw the decision tree?

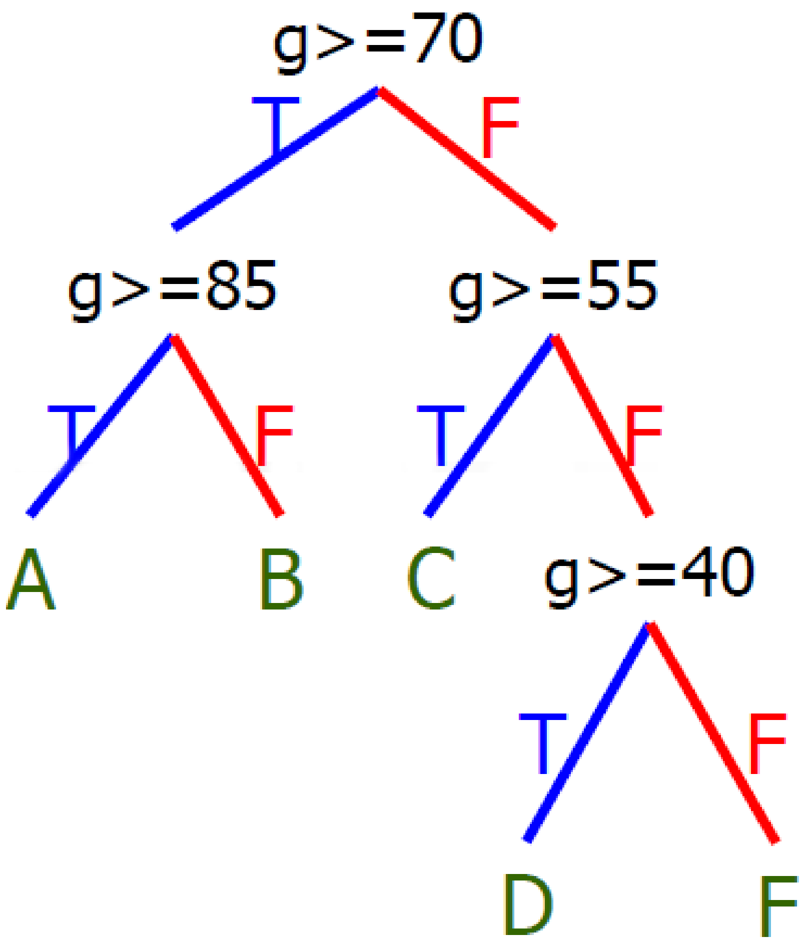
› Nested/multi-way conditions

```
reset countA to countF value to zero
for each student s in class list L do
    g = score of student s
    if g >= 55 then # pass
        if g >= 85 then
            countA = countA + 1
        else
            if g >= 70 then countB = countB + 1
            else countC = countC + 1
    else
        if g >= 40 then countD = countD + 1
        else countF = countF + 1
```

Conditionals

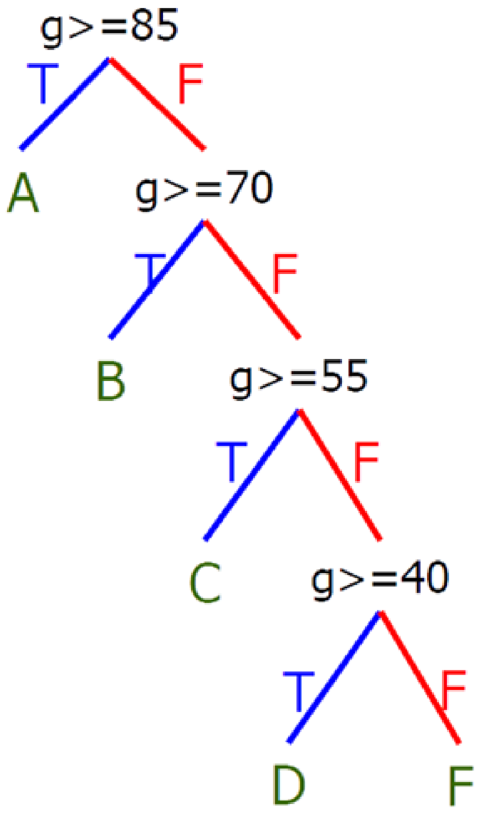
› Conditions can be reordered

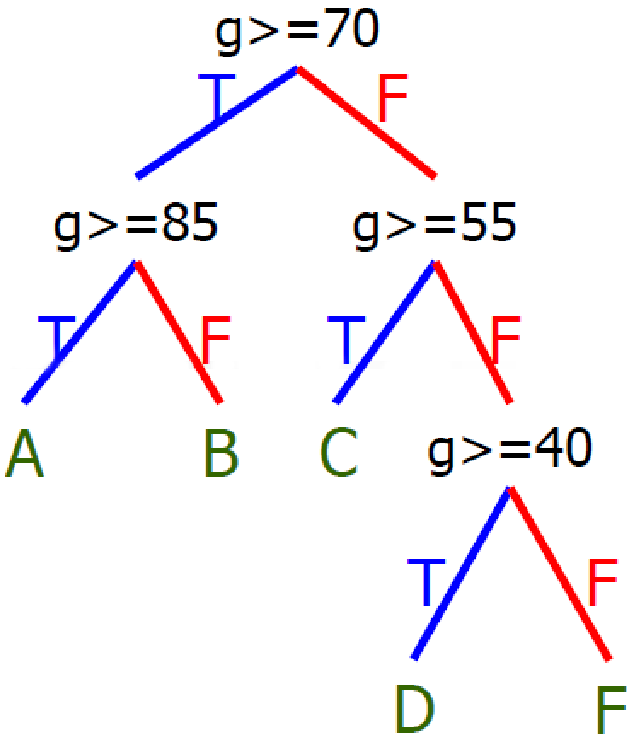
```
reset countA to countF value to zero
for each student s in class list L do
  g = score of student s
  if g >= 70 then
    if g >= 85 then countA = countA + 1
    else countB = countB + 1
  else
    if g >= 55 then
      countC = countC + 1
    else
      if g >= 40 then countD = countD + 1
      else countF = countF + 1
```



Efficiency Consideration

- Average number of times for condition checking
- Assume A=20%, B=50%, C=10%, D=10%, F=10%

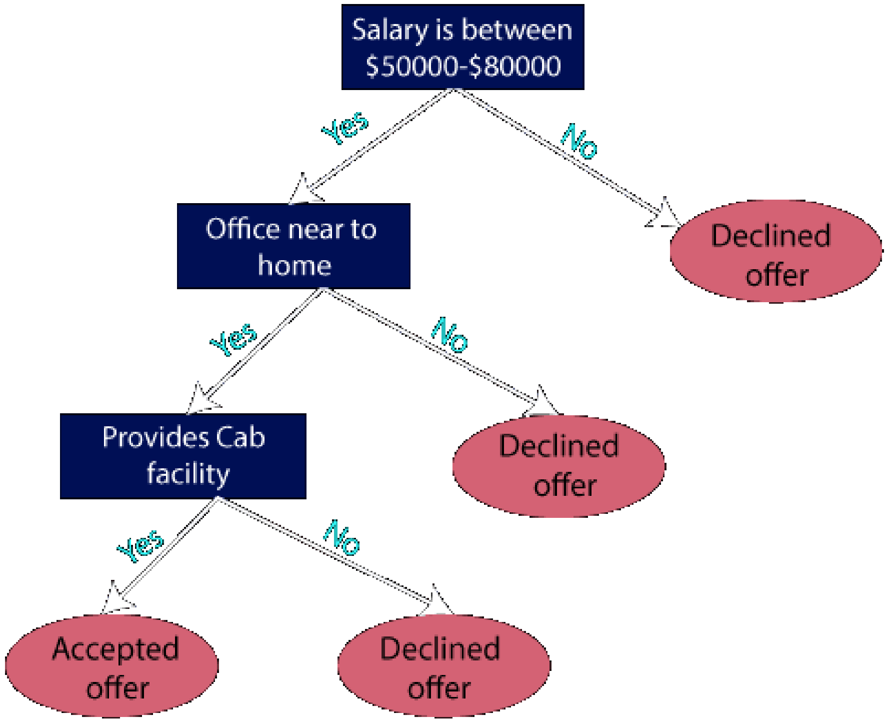




Complex Conditionals

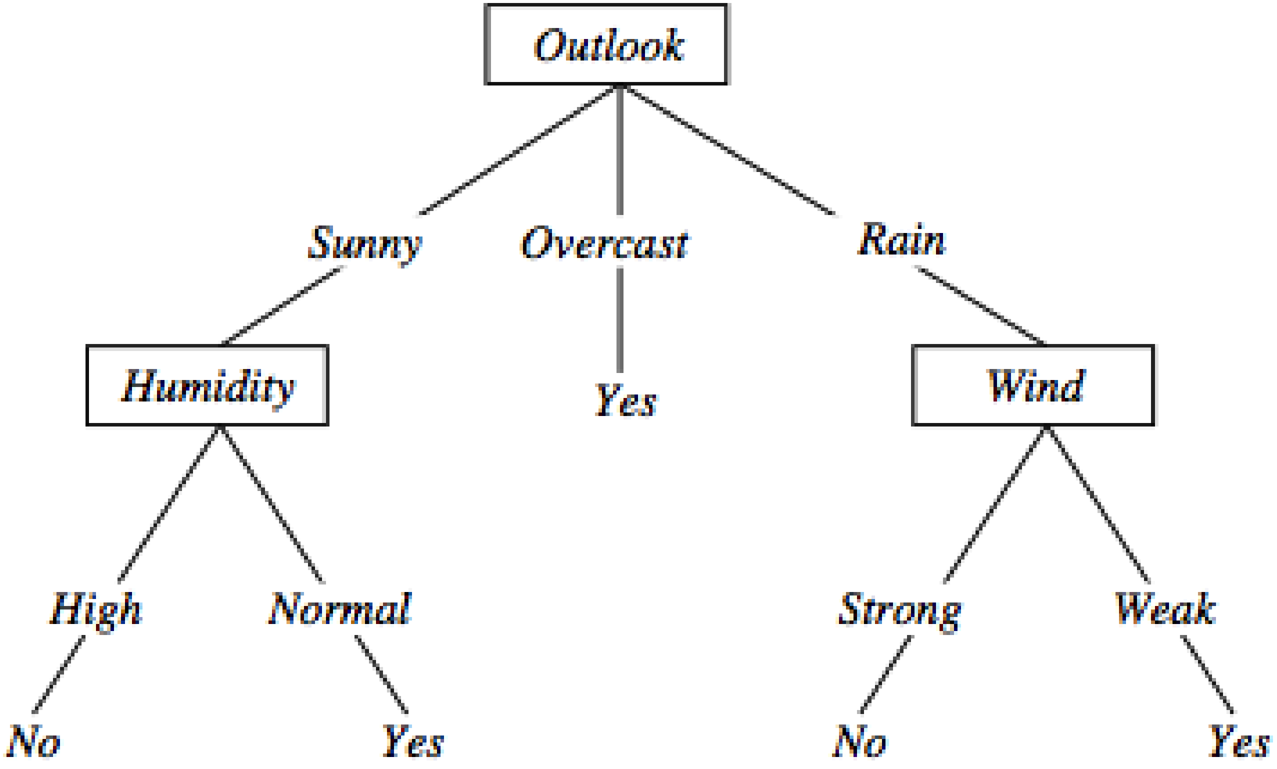
- Nested conditions could best represent complex multi-way decisions
- One very common decision structure is called decision tree, often obtained through *data mining*

```
get salary S, distance D, transportation T
if 50000 <= S <= 80000 then
    if D is near home then
        if T is provided then
            accept offer
        else
            decline offer
    else
        decline offer
else
    decline offer
```



Decision Trees

› Can you express this decision tree on whether to play tennis today using nested conditions?



Decision Trees

- The decision tree is normally derived from a large set of data
- However, not all existing data are consistent with the decision tree, but just most of them
- Not all data attributes are useful in the decision tree

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Sorting Numbers

- Consider that we would like to sort a list of numbers L, e.g., L = [1, 5, 8, 2, 7, 9]
- We need to define the input and output first
  - Input: a list of numbers, L
  - Output: a sorted list M of the numbers
- Possible pseudo-code:

```
let size be the count of numbers in L
set tempList = L and M as an empty list
repeat the following "size" times
    find the smallest number s in tempList
    remove s from tempList
    add s to M
return the new list M
```

Sorting Numbers

- We did not consider in the pseudo-code on how to find the smallest number from a list
  - We just assumed that if it could be solved, we could solve the sorting problem
  - Now we solve this sub-problem of finding the smallest number
    - Input: a list of numbers, L
    - Output: the smallest number s in L
- Possible pseudo-code for sub-problem:

```
set smallest be ∞
for each number n in L
    if n < smallest then set smallest = n
return smallest
```

Sorting Numbers

- Putting them together:

```
let size be count of numbers in L
set tempList = L and M = empty list
repeat the following "size" times
    set s be ∞
    for each number n in tempList
        if n < s then set s = n
    remove s from tempList
    add s to M
return new list M
```

Sorting Numbers

› To have the logic more “modular”:

```
let size be count of numbers in L
set tempList = L and M = empty list
repeat the following “size” times
    s = small(tempList) # find smallest number s in tempList
    remove s from tempList
    add s to M
return new list M
```

```
function small(L):
    set smallest be ∞
    for each number n in L
        if n < smallest then
            set smallest = n
    return smallest
```

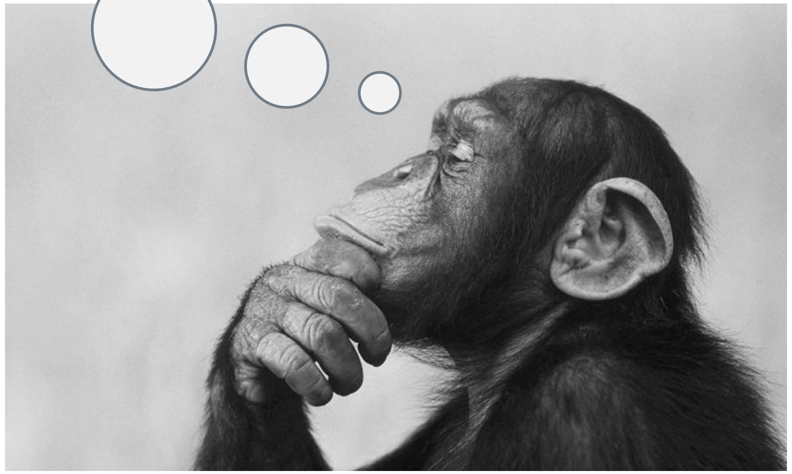
Sorting Numbers

› Another possible pseudo-code:

```
set M as an empty list
for each number n in L
    if M is empty then insert n into M
    if n is smaller than first item in M insert n at the beginning
    if n is larger than last item in M insert n at the end
    find a position p in M such that M[p] < n < M[p+1]
    insert n into M after item p and before item p+1
return M
```

Sorting Numbers

Does the pseudocode work if L contains numbers that are the same? Why?



Summary

› Conditionals
  – List of conditions
  – Chained conditions
  – Nested/multi-way conditions
  – Efficiency Consideration
› Complex Conditionals
  – Decision Trees
› Sorting Numbers