Computational modeling of Webster's problem

Comp 140

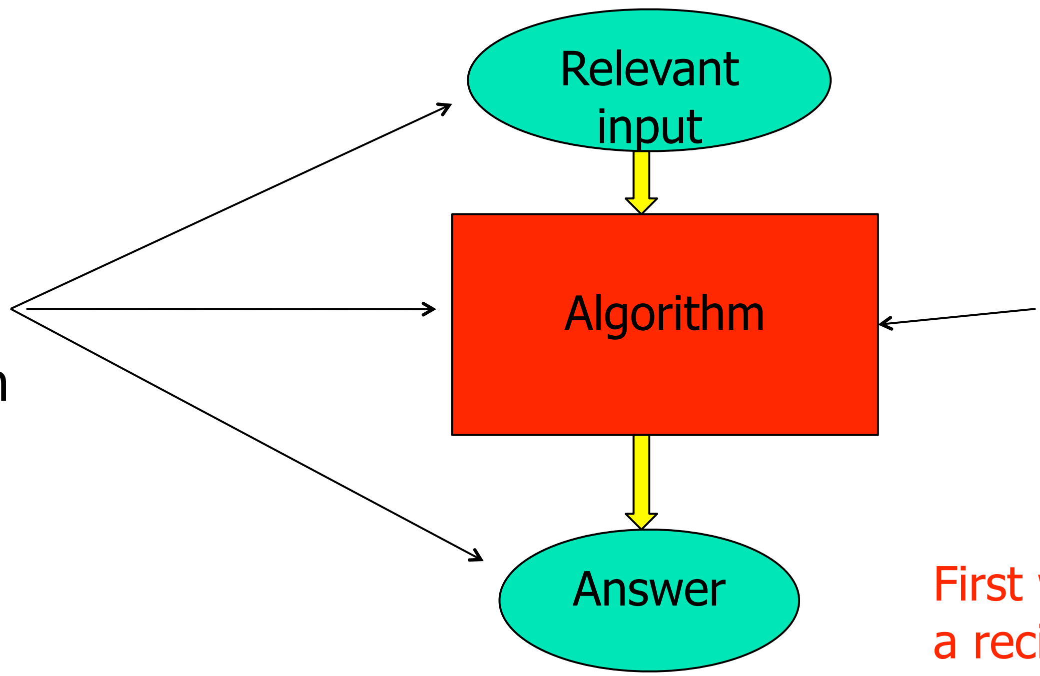
Fall 2008

The “word problem“

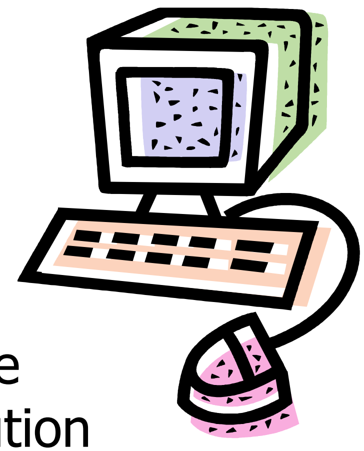
- The devil made a proposition to Daniel Webster. The devil proposed paying Daniel for services in the following way:"On the first day, I will pay you $1,000 early in the morning. At the end of the day, you must pay me a commission of $100. At the end of the day, we will both determine your next day's salary and my commission. I will double what you have earned at the end of the day, but you must double the amount that you pay me. Will you work for me for a month?"

Abstraction and automation

Recipe
Construction



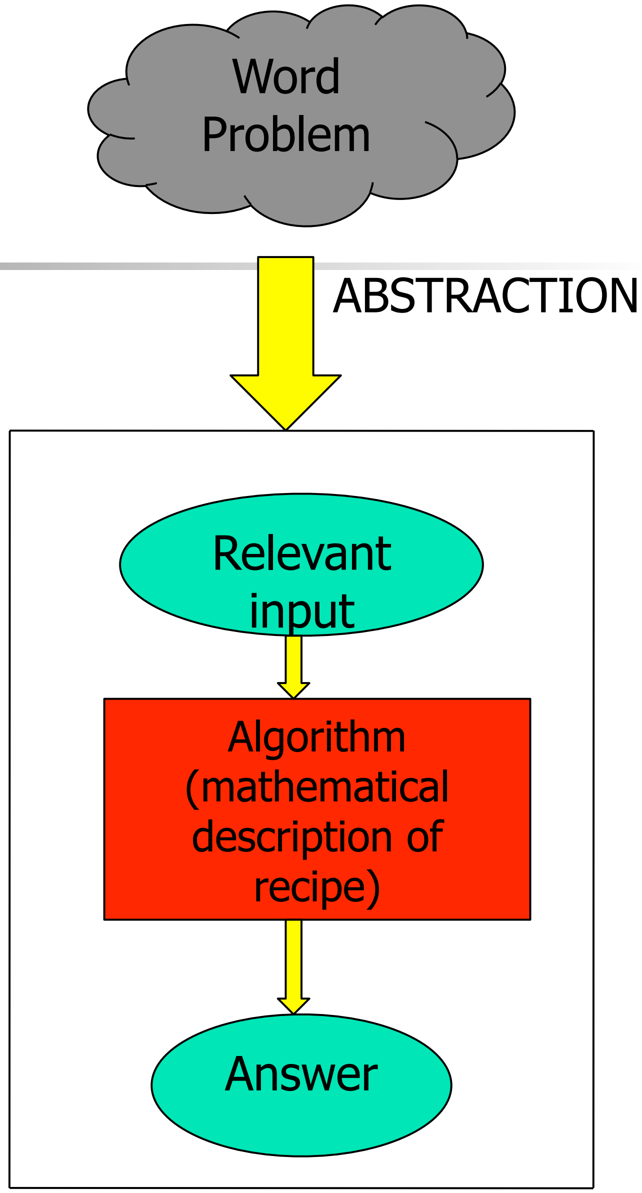
Recipe
Execution
(i.e., the cooking)



First we, as humans, design a recipe.
Then we get the machine to cook it.

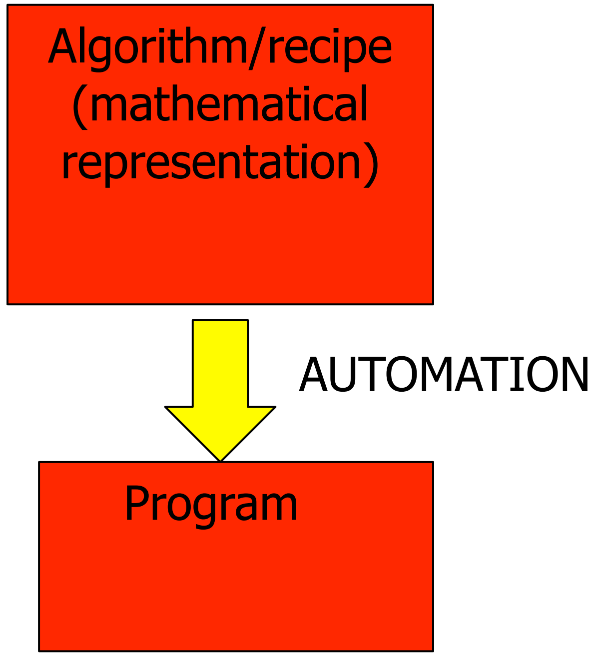
Abstraction

- Identifying the right level at which to model/think about the problem
  - What is to be computed?
  - What are the givens?
  - What is the recipe for computing what we need from the givens?
  - How do we precisely state the recipe to a machine?
- Creative process, requires human ingenuity and thought



Automation

- Communicating a precise recipe to a machine.
- Computational mapping of recipe to data structures and control flow supported by a programming language.
- Translating the mathematical recipe into a program using the chosen computational mapping.



The purpose of the computation

- Should Webster take the devil's deal or not?
- We compute to find the answer to this yes/no question.
- Questions of this form have a name in computer science, they are called decision problems.

Modeling: extracting the relevant pieces of information

- Not all details in the real-world word problem may be necessary for getting to an answer.
- What is the essence of the problem, i.e. what is the relevant information?
- How do we express the essence, the abstraction, in an unambiguous, well-defined manner?

What are the “givens”?

- How the game starts
  - Webster gets a salary of 1000 on day 0.
  - The devil’s commission at the end of day 0 is 100.
- How the game works (from day 0 on)
  - Webster gets a salary at the start of the day.
  - At the end of the day, Webster’s take is his salary minus the commission he pays the devil.
  - The following day, Webster’s salary is double his take from the previous day.
  - The following day, the devil’s commission is twice what he got the previous day.

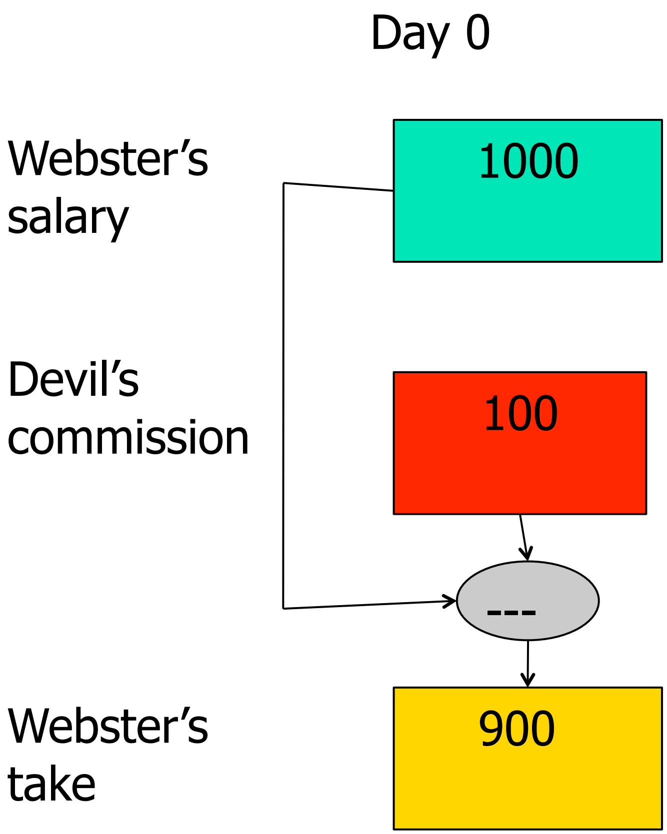
The game illustrated

Day 0

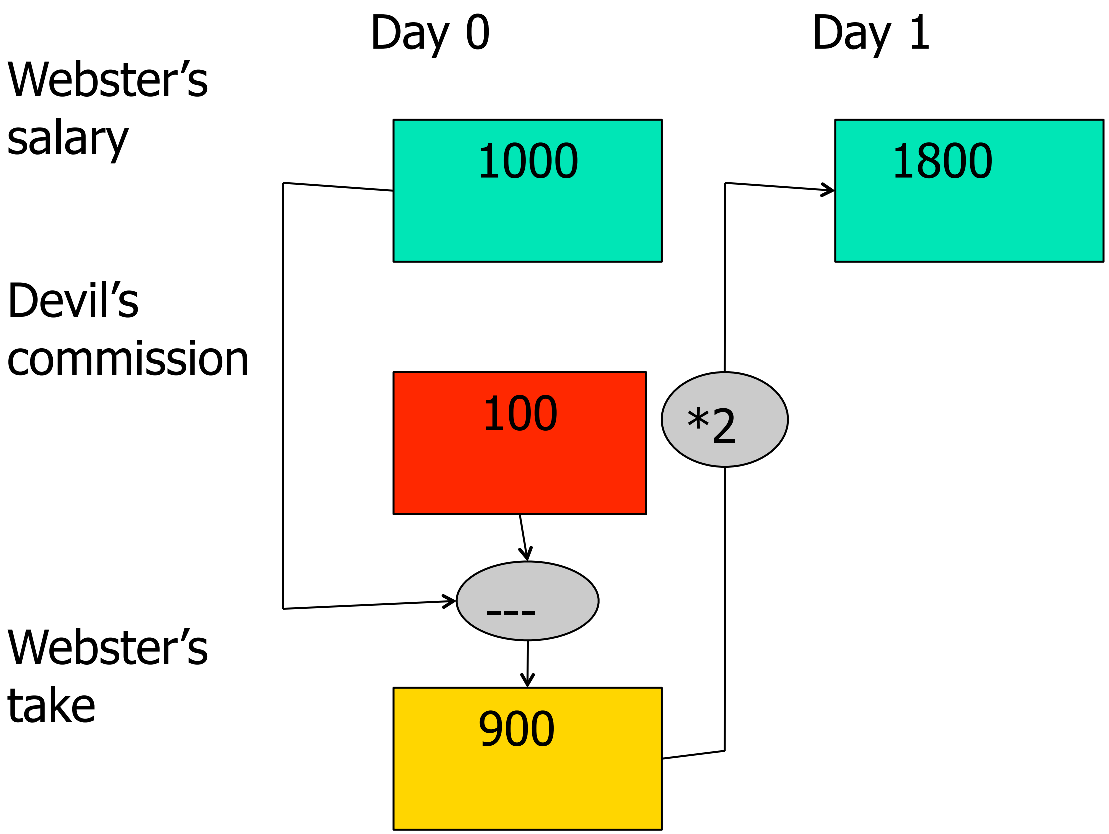
Webster's salary: 1000

Devil's commission: 100

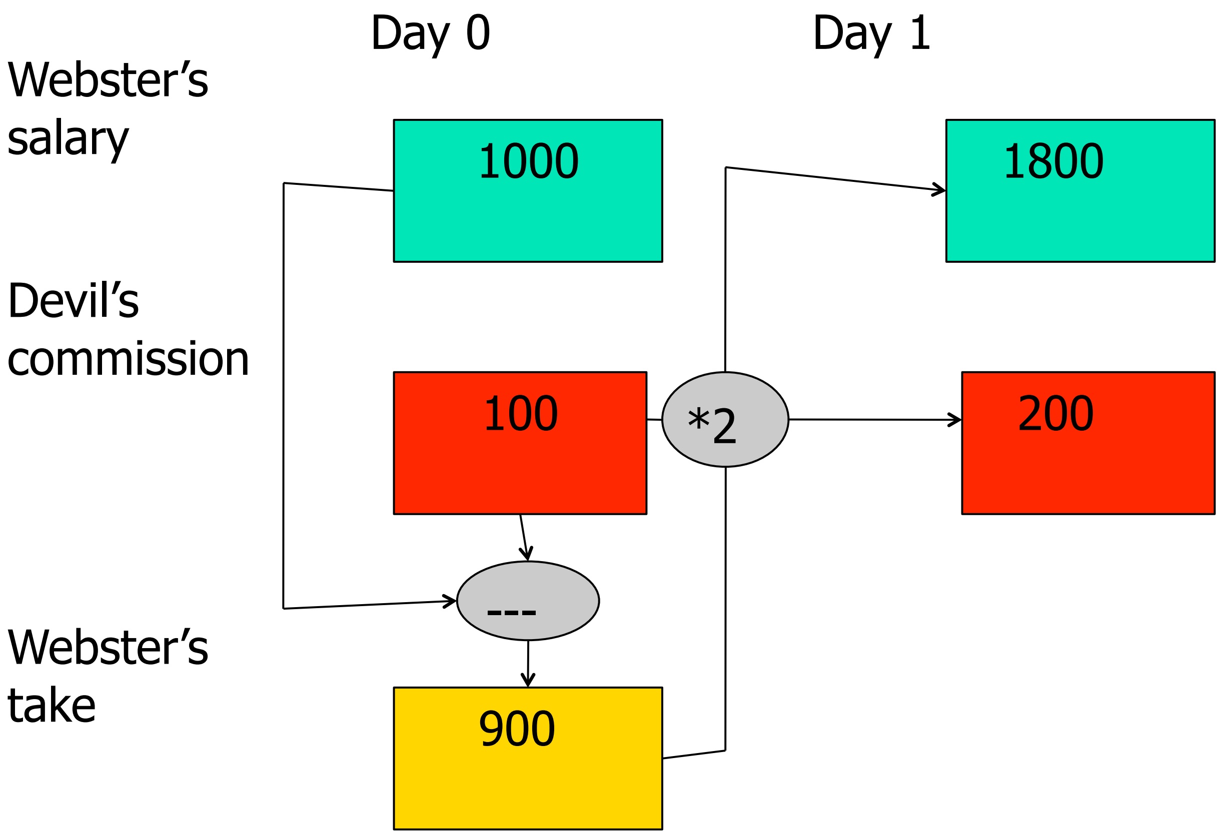
The game continued



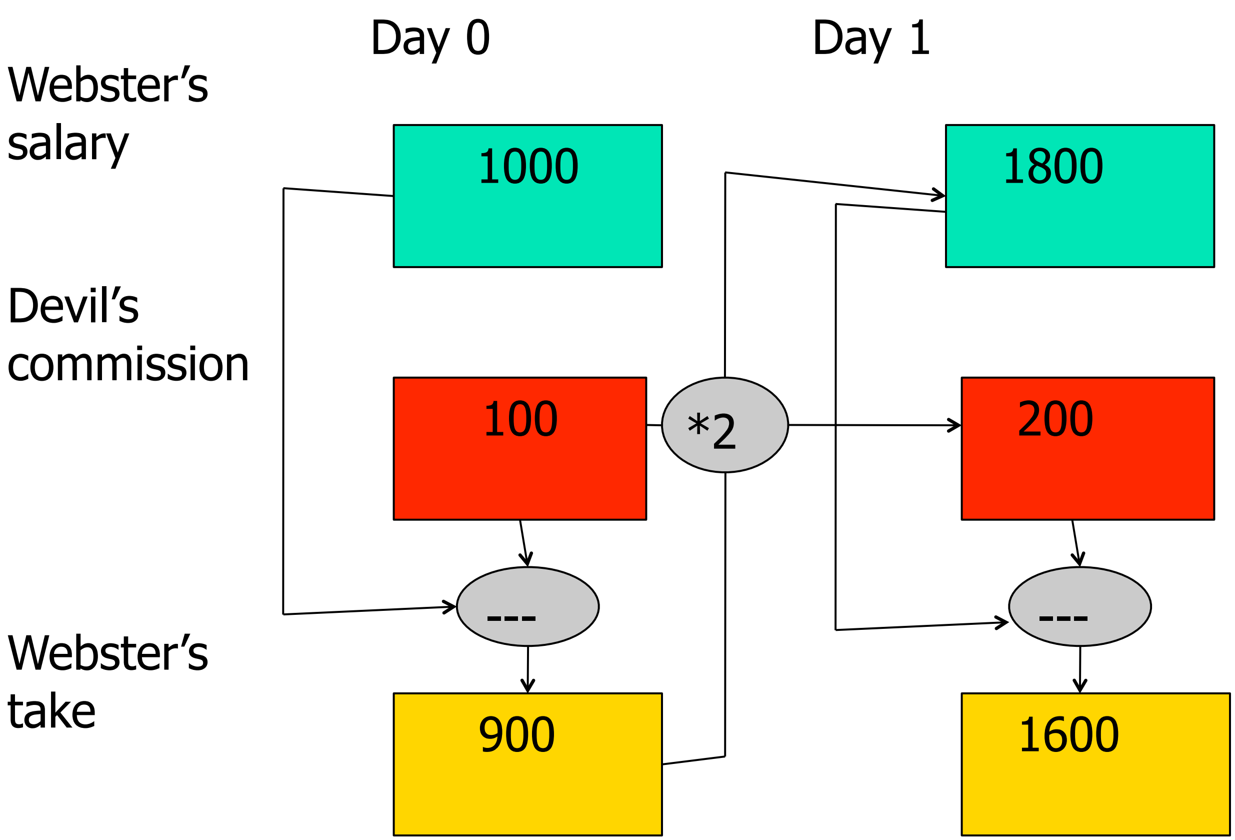
The game continued



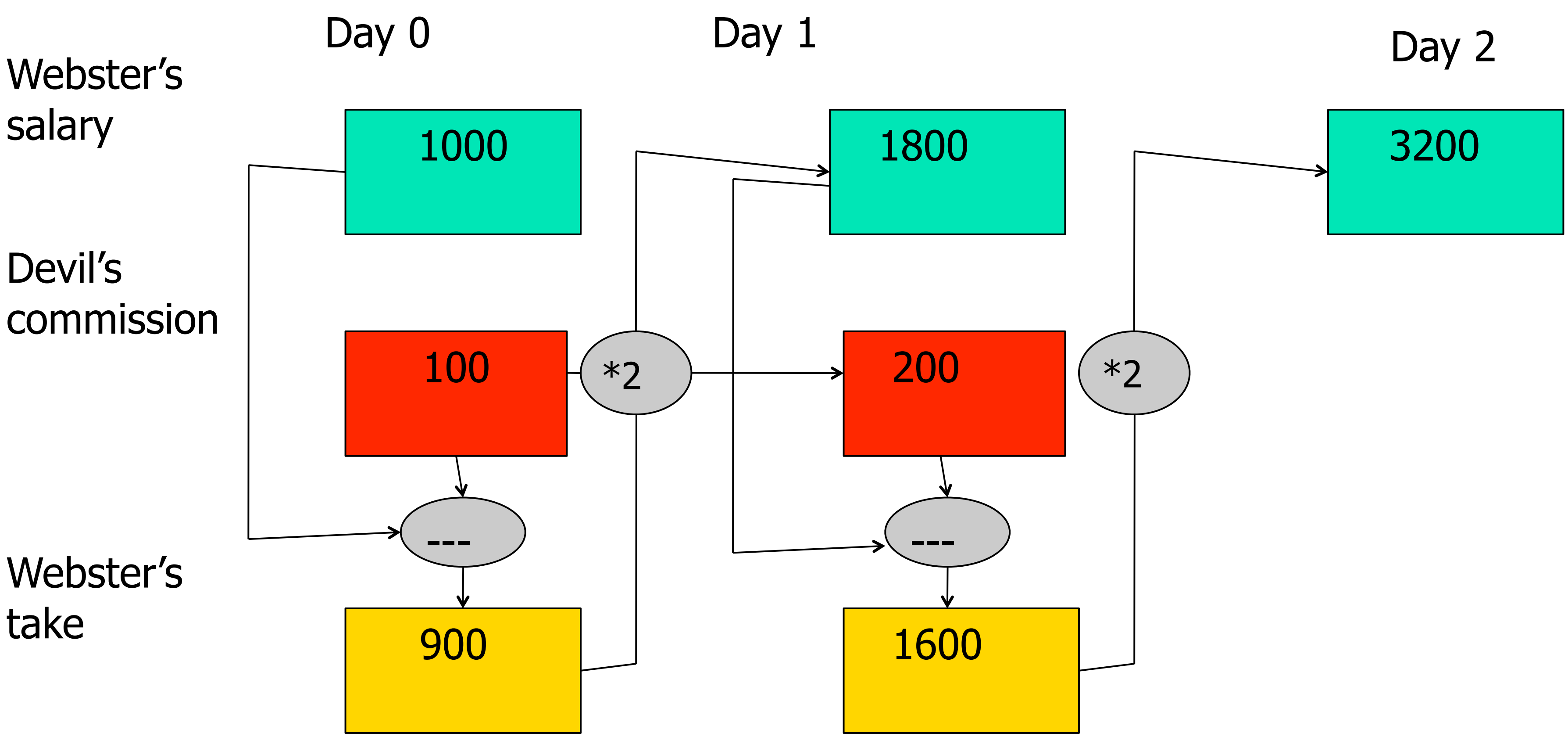
The game continued



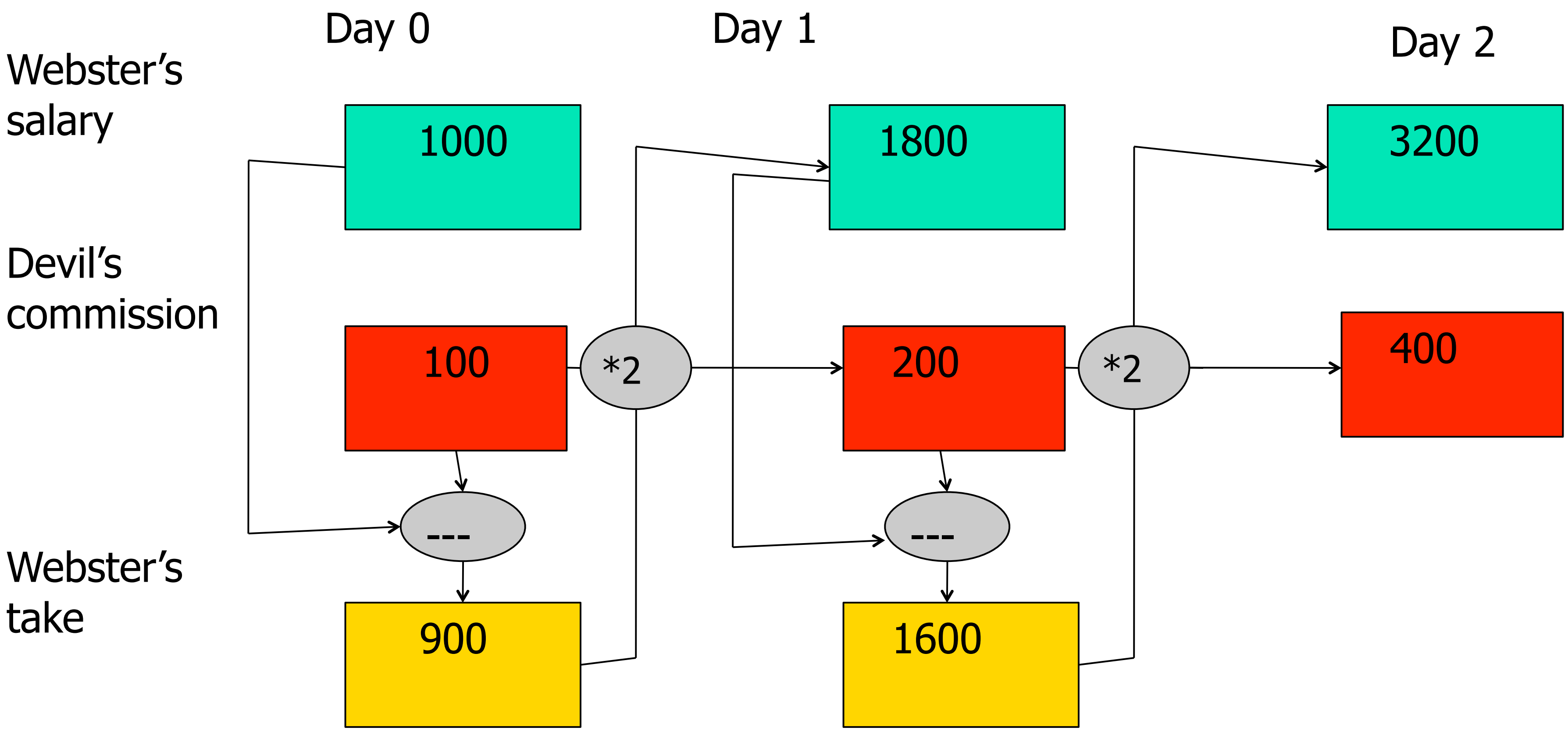
The game continued



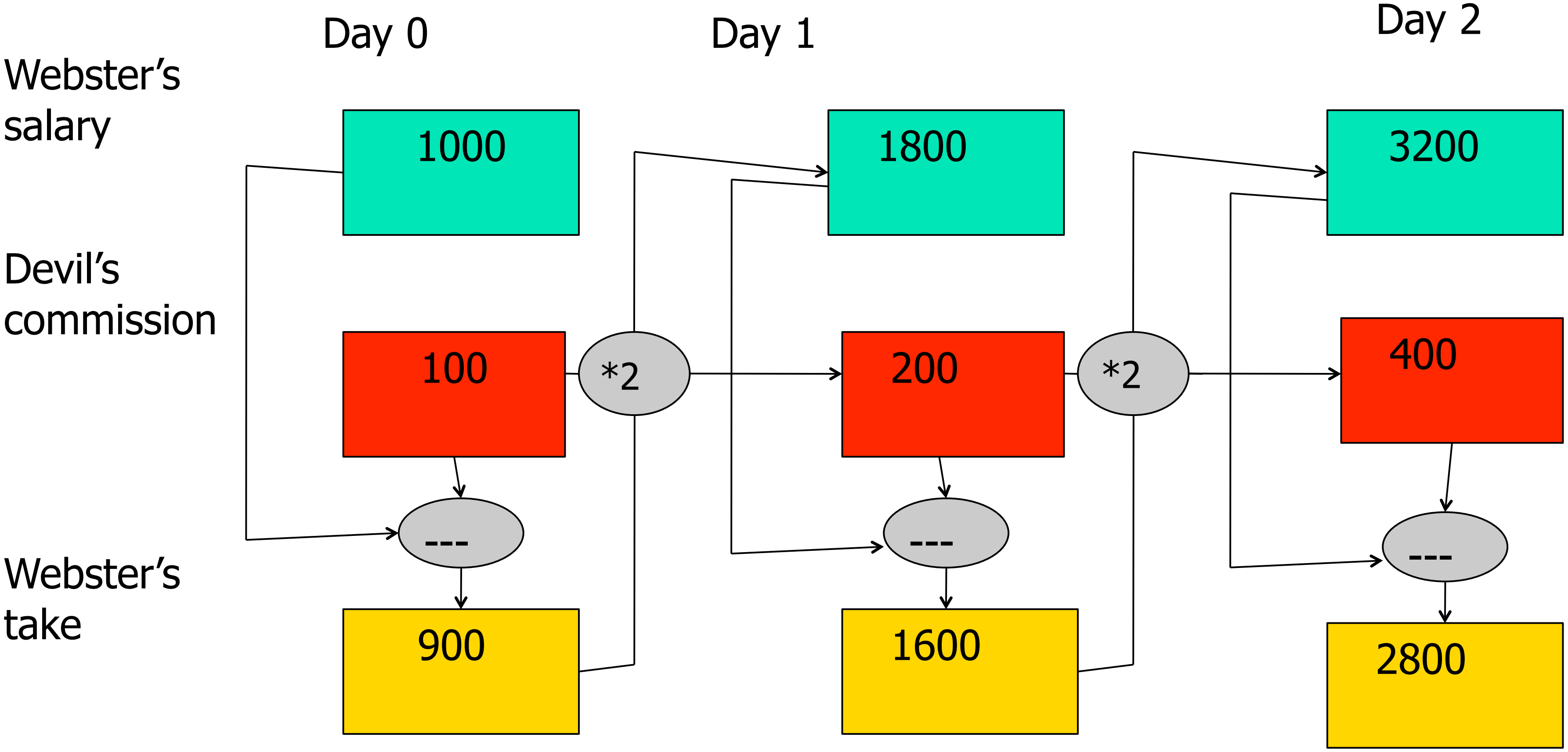
The game continued



The game continued



The game continued



The decision rule

- If Webster's salary on day 30 < 0, then reject the deal.
  - since we count from day 0, "day 30" is actually the 31st day
- Exercise: make another decision rule for this problem

Good notation

- Is key to writing down good recipes
- For this problem
  - we abstracted the English language description of a recipe into a pictorial notation.
  - Pictorial notations are great for communicating with most humans, but not precise enough for computers (ref. CAPTCHAs).
  - we need a more precise representation to communicate with machines

Choosing a language

- ~~Pictures~~
- English
  - Computers are not great at understanding human languages
  - To be fair, neither are humans…(why else do we have the legal system?)
- What else could we use?
  - Hint: how do scientists (natural, social) and engineers communicate their ideas to their peers?

Mathematics: the language of science and computation

- Mathematics is precise and concise.
- Mathematics has well-defined, well-understood operations.
- Mathematics is very expressive, it can represent a lot of real-world phenomena.
- Computers understand Mathematics.
  - at their heart, computers perform simple mathematical functions, e.g. add, subtract.

The main ingredients

Each box in the pictorial notation is an ingredient in the recipe

- Webster's salary:
  - Varies each day
  - So, we will introduce
    - $w_0, w_1, w_2, \ldots, w_{30}$ to denote his salary at the start of day 0, day 1, day 2, …, day 30
- Devil's commission
  - Varies each day
  - So we introduce
    - $d_0, d_1, d_2, \ldots, d_{30}$ to denote his commission at the end of day 0, day 1, day 2, …, day 30

The supporting ingredients

- Webster's take
  - Varies each day
  - Is the difference between Webster's salary at the start of the day and the devil's commission at the end of that day
  - $w_0 - d_0, w_1 - d_1, \ldots, w_{30} - d_{30}$

The full recipe

- $w_0 = 1000$, $d_0 = 100$
- $w_1 = 2(w_0 - d_0) = 1800$, $d_1 = 2d_0 = 200$
- $w_2 = 2(w_1 - d_1) = 3200$, $d_2 = 2d_1 = 400$
- $w_3 = 2(w_2 - d_2) = 5600$, $d_3 = 2d_2 = 800$
- ....
- $w_{30} = 2(w_{29} - d_{29}) = ??$, $d_{30} = 2d_{29} = ??$

The full recipe

- $w_0 = 1000$, $d_0 = 100$
- $w_1 = 2(w_0 - d_0) = 1800$, $d_1 = 2d_0 = 200$
- $w_2 = 2(w_1 - d_1) = 3200$, $d_2 = 2d_1 = 400$
- $w_3 = 2(w_2 - d_2) = 5600$, $d_3 = 2d_2 = 800$
- ....
- $w_{30} = 2(w_{29} - d_{29}) = ??$, $d_{31} = 2d_{30} = ??$

Compact description of recipe

- Webster's salary on day t+1 is twice his take on day t for t=0 through t=29

$$w_{t+1} = 2(w_t - d_t)$$

- Algebra helps us succinctly describe the pattern highlighted in red on slide 24.

The full recipe

- $w_0 = 1000$, $d_0 = 100$
- $w_1 = 2(w_0 - d_0) = 1800$, $d_1 = 2d_0 = 200$
- $w_2 = 2(w_1 - d_1) = 3200$, $d_2 = 2d_1 = 400$
- $w_3 = 2(w_2 - d_2) = 5600$, $d_3 = 2d_2 = 800$
- ….
- $w_{30} = 2(w_{29} - d_{29}) = ??$, $d_{30} = 2d_{29} = ??$

Compact recipe continued

- Devil's commission on day t+1 is twice his commission on day t for t = 0 through t = 29

$$d_{t+1} = 2d_t$$

- Algebra comes to our aid again!

The full compact recipe

$$w_0 = 1000$$
$$d_0 = 100$$
$$for \ t = 0,1,...,29$$
$$w_{t+1} = 2(w_t - d_t)$$
$$d_{t+1} = 2d_t$$

Reject deal if $w_{30} < 0$,

else accept deal

- Repeat these steps for t = 0,1,2,…,29

  t=0: Use w_0 and d_0 to calculate w_1 and d_1

  t=1: Use w_1 and d_1 to calculate w_2 and d_2

  ……

  Use w_{29} and d_{29} to calculate w_{30} and d_{30}

- Apply decision rule

The essence of the problem

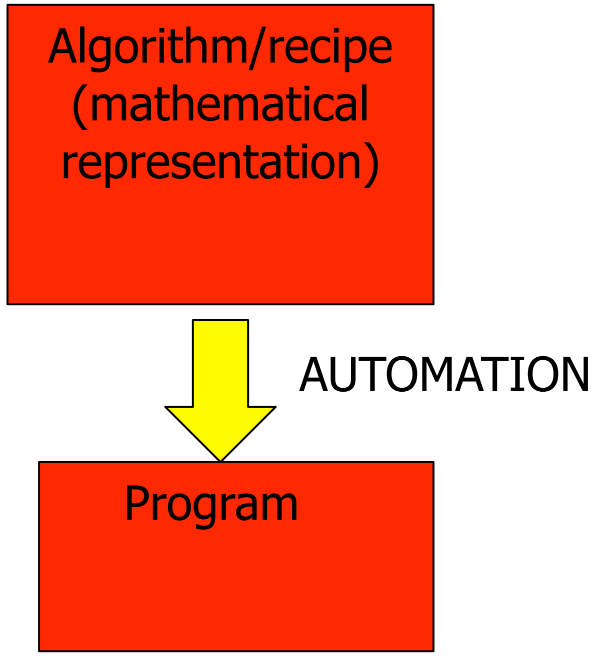
- Only Webster's salary and the Devil's commission are important here. (w and d are the only sequences constructed)
- The key thing is the relationship between the salary and the commission from one day to the next. → use an algebraic equation to represent this relationship succinctly.
- Compute only what you need to make a decision (w_{30}) [note that we only go till t = 29]
- Apply your decision rule to solve problem

$$w_0 = 1000$$
$$d_0 = 100$$
$$\textit{for } \mathrm{t} = 0,1,\ldots,29$$
$$w_{t+1} = 2(w_t - d_t)$$
$$d_{t+1} = 2d_t$$

Reject deal if $w_{30} < 0$, else accept deal

Automation

- Communicating a precise recipe to a machine.
- Computational mapping of recipe to data structures and control flow supported by a programming language.
- Translating the mathematical recipe into a program using the chosen computational mapping.



Mapping ingredient list to computational structures

- Webster's salary
  - is a sequence of numbers, w_0, .., w_{31}
  - naturally maps to Python list

w_0	w_1		w_{31}

- Devil's commission
  - is a sequence of numbers, d_0, …, d_{31}
  - Naturally maps to Python list

d_0	d_1		d_{31}

Note: not the only possible mapping! The great joy of computer science is that there are many good mappings of mathematical structures to computational ones.

A nanotutorial on Python lists

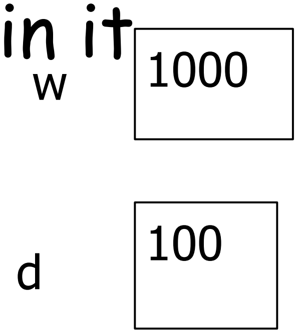
- Create a list
  - >>> numbers = [1,2,3,4,5,6,7,8,9,10]
  - >>> emptyList = []
- Access elements of a list
  - >>> numbers[0]
    - # counting starts at 0
  - >>> numbers[3:6]
    - # first index inclusive, second index exclusive
  - >>> emptyList[0]

Nanotutorial on lists (contd.)

- Append an item to a list
  - >>> numbers.append(11)
- Other list operations
  - >>> dir(numbers)
  - >>> help(numbers)
- Arithmetic operations: * for multiplication, - for subtraction (infix)

Expressing the recipe

- Start the w sequence as a list with a single element 1000 in it
- Start the d sequence as a list with a single element 100 in it



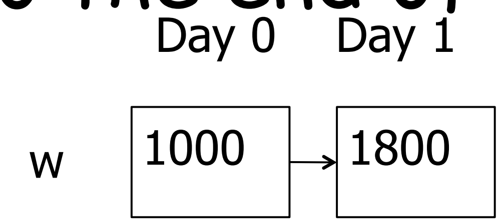
- How do you this in Python?

Python recipe

- w = [1000]
- d = [100]

Compute the next w element

- Calculate twice the difference between w_0 and d_0
- Add it to the end of the list w.



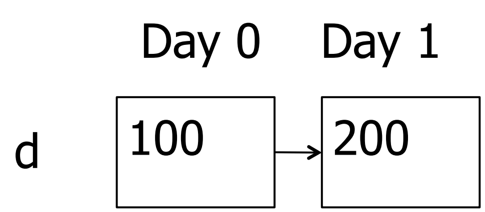
- How do you do this in Python?

Python recipe continued

- w.append(2*(w[0]-d[0]))

Compute the next d element

- Calculate twice d_0
- Add it to the end of list d
  - d.append(2*d[0])



Continuing the computation

- In general, for any day t
  - Calculate twice the difference between w[t] and d[t] and add it to the end of list w.
    - w.append(2*(w[t]-d[t]))
  - Calculate twice d[t] and add it to the end of list d
    - d.append(2*d[t])

How do you tell Python to repeat actions?

- The phrase

```
for x in aList:
     statements
```

  executes the indented statements once for each element in `aList`.
- The function

`range(3)` creates the list `[0, 1, 2]`

The final recipe in Python

```
w = [1000]
d = [100]
for t in range(30):
    w.append(2*(w[t]-d[t]))
    d.append(2*d[t])
```

Seed the two sequences w and d

Go through t values in the order 0,1,2…,29

Extend the sequences w and d each value of t

The process

Day 0

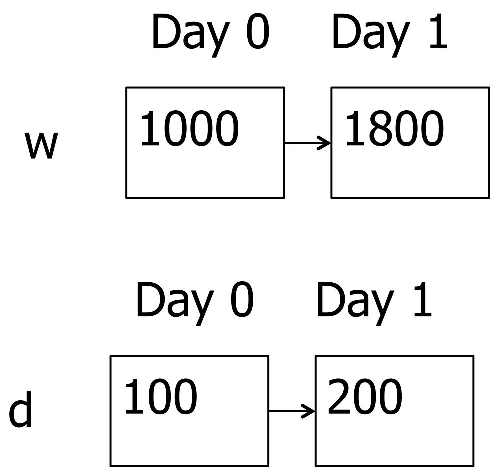
w 1000

Day 0

d 100

```
w = [1000]
d  = [100]
```

The process contd.

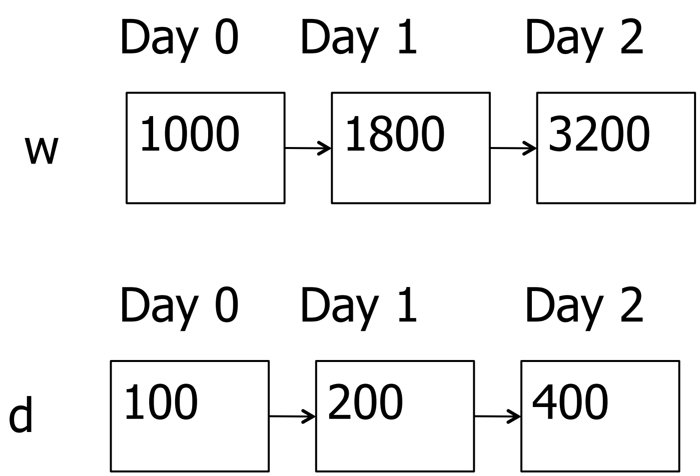


t = 0

```
w.append(2*(w[t]-d[t]))
d.append(2*d[t])
```

Substitute t = 0 here and perform the steps

The process contd.

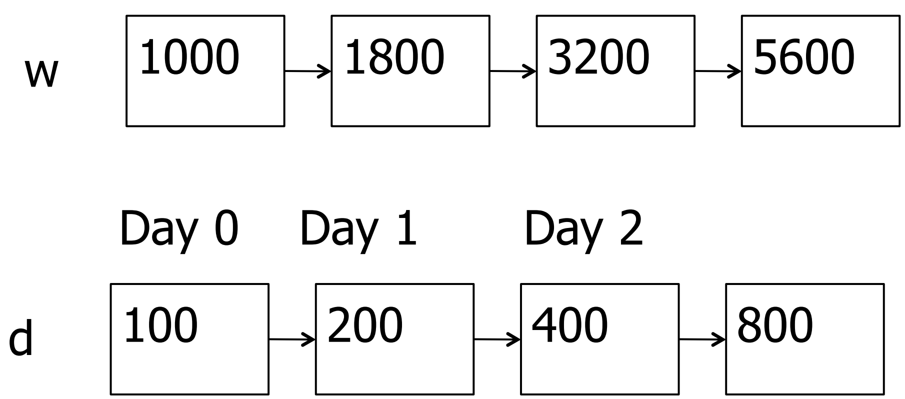


t = 1

```
w.append(2*(w[t]-d[t]))
d.append(2*d[t])
```

Substitute t = 1 here and perform the steps

The process contd.



t = 2

```
w.append(2*(w[t]-d[t]))
d.append(2*d[t])
```

Substitute t = 2 here and perform the steps

The final recipe in Python

```
w = [1000]
d = [100]
for t in range(30):
    print t,w[t],d[t]
    w.append(2*(w[t]-d[t]))
    d.append(2*d[t])
```

To see what Python is computing, ask it to print the new elements added to the sequences (lists) w and d for each day t

Evolution of the recipe

Not mapped to lists

```
w0 = 1000
d0 = 100

w1 = 2*(w0-d0)
d1 = 2*d0

w2 = 2*(w1-d1)
d2 = 2*d1

w3 = 2*(w2-d2)
d3 = 2*d2
…
w30 = 2*(w29-d29)
d30 = 2*d29
```

Verbose computation mapped to lists

```
w = [1000]
d = [100]

w.append(2*(w[0]-d[0]))
d.append(2*d[0])

w.append(2*(w[1]-d[1]))
d.append(2*d[1])
…………
w.append(2*(w[29]-d[29]))
d.append(2*d[29])
```

Concise computation mapped to lists

```
w = [1000]
d = [100]
for t in range(30):
   print t,w[t],d[t]
   w.append(2*(w[t]-d[t]))
   d.append(2*d[t])
```

A specific variation

- Modify the recipe so Webster's counteroffer is to negotiate the length of time he will serve the devil with the $1000 start salary and $100 start commission for the devil.
  - How would you build a recipe for this situation?
  - How would you map the recipe into a Python program to calculate the answer?

More specific variations

- Is there a value for the initial salary that makes it a good deal for Webster? (assume Webster goes back with a counteroffer on his start salary, instead of a straight yes/no answer)
  - How would you build a recipe to calculate this value?
  - How would you map your recipe to a Python program
- Is there a value for the devil's commission that makes it a good deal for Webster? (assume Webster goes back with a counteroffer on the devil's commission, instead of a straight yes/no answer)
  - How would you build a recipe to calculate this value?
  - How would you map your recipe to a Python program

General questions to think about

- Are there other ways to represent Webster's decision problem? Are any of them better than the one suggested here? In what sense are they better?
- Are there other ways to map the mathematical recipe to a computational one? Are they better than the one used here? In what sense are they better?