CSC 171
Introduction to Computer Programming
Lecture 3

Dr. Stephen Bloch
sbloch@adelphi.edu
http://www.adelphi.edu/sbloch/class/171/

Functions with Multiple Arguments

- Write a function named “difference-of-cubes” which takes two numbers (larger and smaller, in that order) and computes the difference of their cubes.
- Follow the design recipe.

Functions with Multiple Arguments

; difference-of-cubes : number (big) number (small) -> number

“Examples of difference-of-cubes:”

(difference-of-cubes 0 0) “should be 0”

(difference-of-cubes 2 0) “should be 8”

(difference-of-cubes 3 2) “should be 19”

Functions with Multiple Arguments

```
; difference-of-cubes : number (big) number (small) ->
    number
(define (difference-of-cubes big small)
    … big … small …)
```

“Examples of difference-of-cubes:”

(difference-of-cubes 0 0) “should be 0”

(difference-of-cubes 2 0) “should be 8”

(difference-of-cubes 3 2) “should be 19”

Functions with Multiple Arguments

```
; difference-of-cubes : number (big) number (small) ->
    number
(define (difference-of-cubes big small)
    (- (* big big big) (* small small small)))
```

“Examples of difference-of-cubes:”

(difference-of-cubes 0 0) “should be 0”

(difference-of-cubes 2 0) “should be 8”

(difference-of-cubes 3 2) “should be 19”

Functions with Multiple Arguments

```
; difference-of-cubes : number (big) number (small) ->
    number
(define (difference-of-cubes big small)
    (- (cube big) (cube small)))
```

“Examples of difference-of-cubes:”

(difference-of-cubes 0 0) “should be 0”

(difference-of-cubes 2 0) “should be 8”

(difference-of-cubes 3 2) “should be 19”

Re-using old functions

- Writing, testing & debugging is a pain.
- Once you've written (tested & debugged) a function, get as much mileage from it as you can.
- Re-use it to help you write other functions more simply and briefly.
- A good programmer is lazy!

Re-using old functions

- To re-use a previously-defined function, it needs to be defined in the same Definitions window, *before the function that uses it*.
- I recommend doing a whole homework assignment in one Definitions window, so you can re-use easily.

Exercise

- Define a function “total-minutes” which takes in a number of hours and a number of minutes, in that order, and returns the total number of minutes.
- Ex. (total-minutes 1 30) “should be 90”
- Follow the design recipe.

Recall: kinds of information

- Numbers
  - Integers, e.g. 3, -29, 12870985701928745
  - Fractions, e.g. 3/4, 23897/38904
  - “floating-point” numbers, e.g. #i3.1415926
- Booleans: true and false
- Strings, e.g. “should be 7”, “Dr. Bloch”
- Pictures

Contracts for predefined functions

- + takes two or more numbers & returns a number
- / takes two numbers & returns a number
- **sqrt** takes one number & returns a number
- **cos** takes one number & returns a number
- etc. etc.

Can’t use *any* function correctly without knowing its contract!

Contracts for more predefined functions

- = takes two numbers and returns a Boolean
- >, <, >=, <= each takes two numbers and returns a Boolean
- **even?** takes in a number and returns a Boolean (Convention: names ending in “?” mean the function returns a Boolean)
- **not** takes in a Boolean and returns a Boolean
- **and, or** each takes in two or more Booleans and returns a Boolean

Shorter notation for contracts

- + : number number … -> number
- / : number (numerator) number (denominator) -> number
- **sqrt** : number -> number
- **cos** : number -> number
- **difference-of-cubes** : number (big) number (small) -> number
- = : number number -> boolean
- **and** : boolean boolean ... -> boolean
- Note: when there are several parameters, and it makes a difference which one is which, it helps to give them names in the contract.

Example

; The “under-100?” function tells whether a number is less than 100.

; under-100? : number -> boolean

Subranges of numbers

- Input is numbers, but divided into two subranges:
  - Anything under 100, and
  - Anything 100 or more

Choosing examples

- If output is a Boolean, make sure there's at least
  - One example with right answer true
  - One example with right answer false
- If input is ranges of numbers, make sure there's at least
  - One example in each subrange
  - One example at each boundary

Example

; The “under-100?” function tells whether a number is less than 100.

; under-100? : number -> boolean

“Examples of under-100?:”

(under-100? 73) “should be true”

(under-100? 100) “should be false”

(under-100? 101) “should be false”

Example

```
; The “under-100?” function tells whether a number is less than
    100.

; under-100? : number -> boolean

(define (under-100? num)
    ... num ... )

“Examples of under-100?:”

(under-100? 73) “should be true”

(under-100? 100) “should be false”

(under-100? 101) “should be false”
```

Example

```
; The “under-100?” function tells whether a number is less than
    100.

; under-100? : number -> boolean

(define (under-100? num)
    (< num 100))

“Examples of under-100?:”

(under-100? 73) “should be true”

(under-100? 100) “should be false”

(under-100? 101) “should be false”
```

Homework problem

- The function “can-vote?” takes in a person’s age (a number of years) and returns a true or false indicating whether that person is at least 18.
- Follow the design recipe.

Homework problem

- The function “can-vote-but-not-drink?” takes in a person’s age (a number of years) and returns a true or false indicating whether that person is at least 18 but under 21.
- Follow the design recipe.