Organizational and HW Issues

- No more lab 2 (Th 7:30-8:20)
- Subscribe to **`cs2010@cs.utah.edu`**
- Consulting hours now on the web page
- Handin button submits whatever is in the DrScheme window
  - File name (or whether it's saved) doesn't matter
- To define a constant:

`(define me`  **`)`**

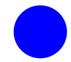
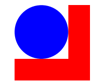
How to Design Programs

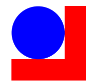
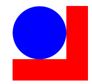
Computation versus Programming

- Last time, we talked about computation

Computation versus Programming

- Last time, we talked about computation

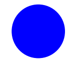
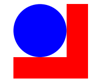
(image=? (image+  ) )

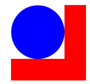
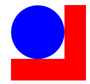
→ (image=?  )

→ true

Computation versus Programming

- Last time, we talked about computation

`(image=? (image+`  `)` `)`

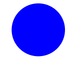
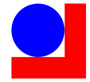
$\rightarrow$ `(image=?`  `)`

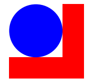
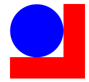
$\rightarrow$ `true`

- Programming?

Computation versus Programming

- Last time, we talked about computation

(image=? (image+  ) )

→ (image=?  )

→ true

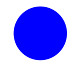
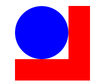
- Programming?

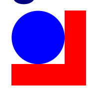
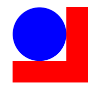
Write an anonymizer...

```
(define (anonymize i)
  (offset-masked-image+
   i 0 0
   (solid-dot (image-width i) (image-height i)
              'black)
   (solid-dot (image-width i) (image-height i)
              'blue)))
```

Computation versus Programming

- Last time, we talked about computation

(image=? (image+  ) )

→ (image=?  )

→ true

- Programming?

Write an anonymizer...

```
(define (anonymize i)
  (offset-masked-image+
   i 0 0
   (solid-dot (image-width i) (image-height i)
              'black)
   (solid-dot (image-width i) (image-height i)
              'blue)))
```

We somehow wrote the function in one big, creative chunk.

Programming

Today: ***How to Design Programs***

- Programming always requires creativity
- But a design rules can guide and focus creativity

Programming

Today: ***How to Design Programs***

- Programming always requires creativity
- But a design rules can guide and focus creativity

Analogous to rules for composing music:
scales, chords, counterpoint, rhythms, etc.

Programming

Today: ***How to Design Programs***

- Programming always requires creativity
- But a design rules can guide and focus creativity

Analogous to rules for composing music:
scales, chords, counterpoint, rhythms, etc.

Language syntax is like musical notation.
You need a notation, but notation alone gets you nowhere.

The Design Recipe

- We'll start with a simple recipe
- As the course progresses, we'll expand the recipe

Design Recipe I

Data

- Understand the input data: `num`, `bool`, `sym`, or `image`

Contract, Purpose, and Header

- Describe (but don’t write) the function

Examples

- Show what will happen when the function is done

Body

- The most creative step: implement the function body

Test

- Run the examples

Design Recipe I

Data

- Understand the input data: **`num`**, **`bool`**, **`sym`**, or **`image`**

Contract, Purpose, and Header

- Describe (but don’t write) the function

Examples

- Show what will happen when the function is done

Body

- The most creative step: implement the function body

Test

- Run the examples

Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `sym`
- Faces ➡ `image`
- Wages ➡ `num`
- ...

Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ **`num`**
- Grocery items ➡ **`sym`**
- Faces ➡ **`image`**
- Wages ➡ **`num`**
- ...

Handin artifact: **none** for now

Design Recipe I

Data

- Understand the input data: `num`, `bool`, `sym`, or `image`

Contract, Purpose, and Header

- Describe (but don’t write) the function

Examples

- Show what will happen when the function is done

Body

- The most creative step: implement the function body

Test

- Run the examples

Contract, Purpose, and Header

Contract

Describes input(s) and output data

- `f2c : num -> num`
- `is-milk? : sym -> bool`
- `wearing-glasses? : image image image -> bool`
- `netpay : num -> num`

Contract, Purpose, and Header

Contract

Describes input(s) and output data

- **`f2c : num -> num`**
- **`is-milk? : sym -> bool`**
- **`wearing-glasses? : image image image -> bool`**
- **`netpay : num -> num`**

Handin artifact: a comment

```
; f2c : num -> num
; is-milk? : sym -> bool
```

Contract, Purpose, and Header

Purpose

Describes, in English, what the function will do

- Converts F-degrees `f` to C-degrees
- Checks whether `s` is a symbol for milk
- Checks whether `p2` is `p1` wearing glasses `g`
- Computes net pay (less taxes) for `n` hours worked

Contract, Purpose, and Header

Purpose

Describes, in English, what the function will do

- Converts F-degrees `f` to C-degrees
- Checks whether `s` is a symbol for milk
- Checks whether `p2` is `p1` wearing glasses `g`
- Computes net pay (less taxes) for `n` hours worked

Handin artifact: a comment after the contract

```
; f2c : num -> num
; Converts F-degrees f to C-degrees
```

Contract, Purpose, and Header

Header

Starts the function using variables that are metioned in purpose

- `(define (f2c f) ....)`
- `(define (is-milk? s) ....)`
- `(define (wearing-glasses? p1 p2 g) ....)`
- `(define (netpay n) ....)`

Contract, Purpose, and Header

Header

Starts the function using variables that are metioned in purpose

- `(define (f2c f) ....)`
- `(define (is-milk? s) ....)`
- `(define (wearing-glasses? p1 p2 g) ....)`
- `(define (netpay n) ....)`

Check: function name and variable count match contract

Contract, Purpose, and Header

Header

Starts the function using variables that are metioned in purpose

- `(define (f2c f) ....)`
- `(define (is-milk? s) ....)`
- `(define (wearing-glasses? p1 p2 g) ....)`
- `(define (netpay n) ....)`

Check: function name and variable count match contract

Handin artifact: as above, but absorbed into implementation

```
; f2c : num -> num
; Converts F-degrees f to C-degrees
(define (f2c f) ....)
```

Design Recipe I

Data

- Understand the input data: `num`, `bool`, `sym`, or `image`

Contract, Purpose, and Header

- Describe (but don't write) the function

Examples

- Show what will happen when the function is done

Body

- The most creative step: implement the function body

Test

- Run the examples

Examples

Show example function calls an result

```
(f2c 32) "should be" 0
(f2c 212) "should be" 100

(is-milk? ’milk) "should be" true
(is-milk? ’apple) "should be" false
```

Examples

Show example function calls an result

```
(f2c 32) "should be" 0
(f2c 212) "should be" 100

(is-milk? 'milk) "should be" true
(is-milk? 'apple) "should be" false
```

Check: function name, argument count and types match contract

Examples

Show example function calls an result

```
(f2c 32) "should be" 0
(f2c 212) "should be" 100

(is-milk? ’milk) "should be" true
(is-milk? ’apple) "should be" false
```

Check: function name, argument count and types match contract

Handin artifact: as above, after header/body

```
; f2c : num -> num
; Converts F-degrees f to C-degrees
(define (f2c f) ....)
(f2c 32) "should be" 0
(f2c 212) "should be" 100
```

Design Recipe I

Data

- Understand the input data: `num`, `bool`, `sym`, or `image`

Contract, Purpose, and Header

- Describe (but don't write) the function

Examples

- Show what will happen when the function is done

Body

- The most creative step: implement the function body

Test

- Run the examples

Body

Fill in the body under the header

```
(define (f2c f)
  (* (- f 32) 5/9))

(define (is-milk? s)
  (symbol=? s ’milk))
```

Body

Fill in the body under the header

```
(define (f2c f)
  (* (- f 32) 5/9))

(define (is-milk? s)
  (symbol=? s 'milk))
```

Handin artifact: complete at this point

```
; f2c : num -> num
; Converts F-degrees f to C-degrees
(define (f2c f)
  (* (- f 32) 5/9))
(f2c 32) "should be" 0
(f2c 212) "should be" 100
```

Design Recipe I

Data

- Understand the input data: `num`, `bool`, `sym`, or `image`

Contract, Purpose, and Header

- Describe (but don't write) the function

Examples

- Show what will happen when the function is done

Body

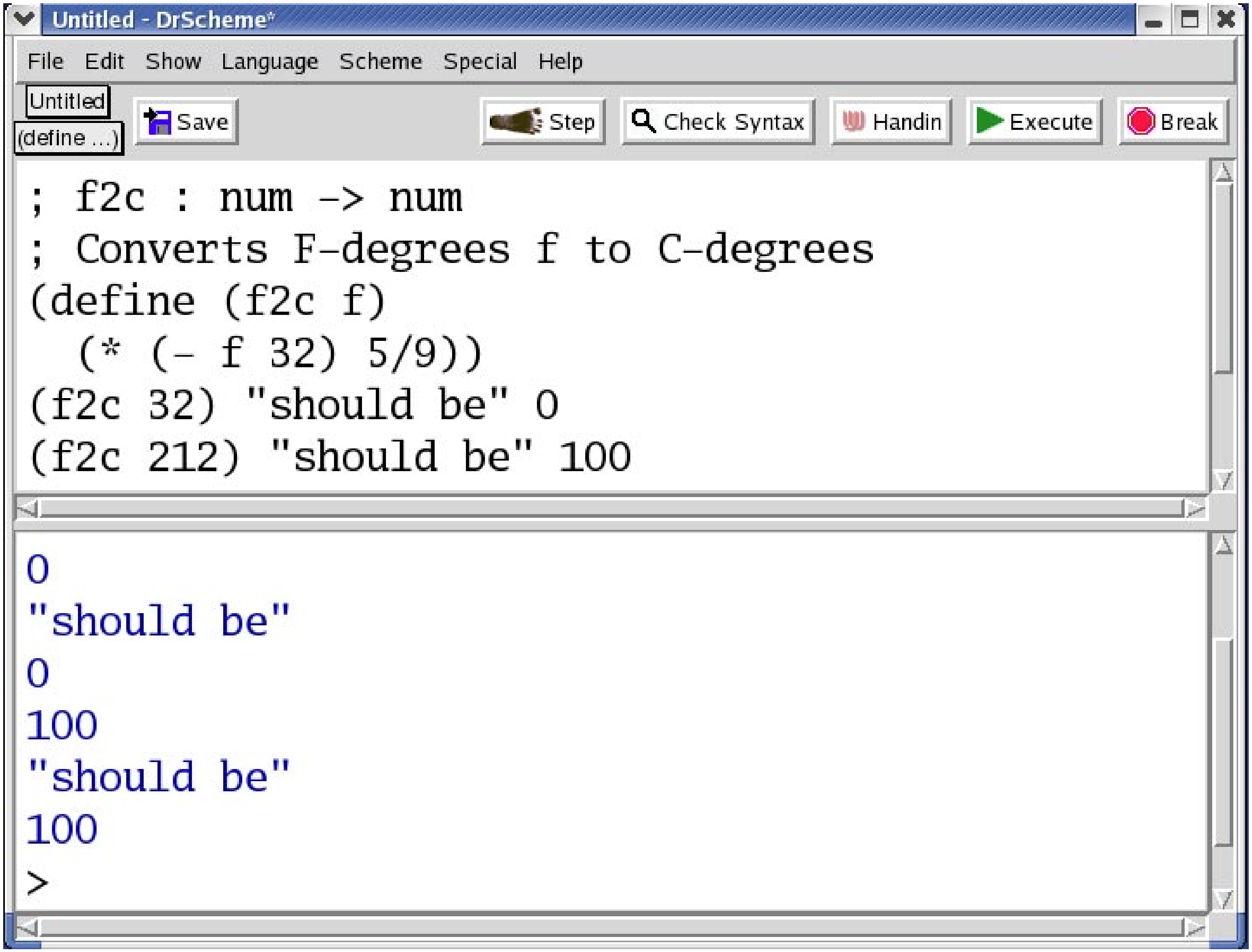
- The most creative step: implement the function body

Test

- Run the examples

Test

Click **Execute**- examples serve as tests



Design Recipe - Each Step Has a Purpose

Data

- Shape of input data will drive the implementation

Contract, Purpose, and Header

- Provides a first-level understanding of the function

Examples

- Gives a deeper understanding and exposes specification issues

Body

- The implementation is the whole point

Test

- Evidence that it works

Design Recipe FAQ

- Do I have to use the recipe when the function seems obvious?
  - **Yes.**

Design Recipe FAQ

- Do I have to use the recipe when the function seems obvious?
  - **Yes.**
- Will my grade suffer if I don’t handin recipe artifacts?
  - **Yes,** except for HW 1

Design Recipe FAQ

- Do I have to use the recipe when the function seems obvious?
  - **Yes.**
- Will my grade suffer if I don’t handin recipe artifacts?
  - **Yes,** except for HW 1
- Isn’t the recipe just a lot of obnoxious busy work?
  - **No.** It’s a training exercise.

As programs become more complex in the next few weeks, the design recipe will prove more helpful.

If you don’t learn to use the recipe now, you’ll be stuck having to learn both the recipe and other concepts later on.