## Computation versus Programming

Last time, we talked about **computation**

## Computation versus Programming

Last time, we talked about **computation**

```
(+ 1 (* 2 3)) → (+ 1 6) → 7
```

## Computation versus Programming

Last time, we talked about **computation**

`(+ 1 (* 2 3))` → `(+ 1 6)` → `7`

**Programming**?

## Computation versus Programming

Last time, we talked about **computation**

```
(+ 1 (* 2 3)) → (+ 1 6) → 7
```

**Programming**?

Make a wanted poster... ➡

```
(define (maybe-wanted who wanted-who)
  (cond
   [(image=? who wanted-who)
    (above (text "WANTED" 32 "black") who)]
   [else
    who]))
```

## Computation versus Programming

Last time, we talked about **computation**

`(+ 1 (* 2 3))` → `(+ 1 6)` → `7`

**Programming**?

Make a wanted poster... ➡

```
(define (maybe-wanted who wanted-who)
  (cond
   [(image=? who wanted-who)
    (above (text "WANTED" 32 "black") who)]
   [else
    who]))
```

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

- 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`, `string`, or `image`

## Signature, 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`, `string`, or `image`

## Signature, 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

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `string`

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `string`
- Faces

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `string`
- Faces ➡ `image`

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `string`
- Faces ➡ `image`
- Wages

## Data

Choose a representation suitable for the function input

- Fahrenheit degrees ➡ `num`
- Grocery items ➡ `string`
- Faces ➡ `image`
- Wages ➡ `num`
- …

## Data

Choose a representation suitable for the function input

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

Handin artifact: **none** for now

# Design Recipe I

## Data

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

## Signature, 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

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

- `f2c`

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

- `f2c : num -> num`

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

- `f2c : num -> num`
- `is-milk?`

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

- `f2c : num -> num`
- `is-milk? : string -> bool`

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

- `f2c : num -> num`
- `is-milk? : string -> bool`
- `wearing-glasses?`

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

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

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

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

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

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

## Signature, Purpose, and Header

***Signature***

Describes input(s) and output data

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

Handin artifact: a comment

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

## Signature, Purpose, and Header

***Purpose***

Describes, in English, what the function will do

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

## Signature, Purpose, and Header

***Purpose***

Describes, in English, what the function will do

- Converts F-degrees `f` to C-degrees
- Checks whether `s` is a string 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 signature

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

## Signature, 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) ....)`

## Signature, 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 signature

## Signature, 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 signature

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`, `string`, or `image`

### Signature, 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

```
(check-expect (f2c 32) 0)
(check-expect (f2c 212) 100)

(check-expect (is-milk? "milk") #true)
(check-expect (is-milk? "apple") #false)
```

## Examples

Show example function calls an result

```
(check-expect (f2c 32) 0)
(check-expect (f2c 212) 100)

(check-expect (is-milk? "milk") #true)
(check-expect (is-milk? "apple") #false)
```

Check: function name, argument count and types match signature

## Examples

Show example function calls an result

```
(check-expect (f2c 32) 0)
(check-expect (f2c 212) 100)

(check-expect (is-milk? "milk") #true)
(check-expect (is-milk? "apple") #false)
```

Check: function name, argument count and types match signature

Handin artifact: as above, after header/body

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

# Design Recipe I

## Data

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

## Signature, 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)
  (string=? s "milk"))
```

## Body

Fill in the body under the header

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

(define (is-milk? s)
  (string=? 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))
(check-expect (f2c 32) 0)
(check-expect (f2c 212) 100)
```

# Design Recipe I

## Data

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

## Signature, Purpose, and Header

- Describe (but don't write) the function

## Examples

- Show what will happen when the function is done

## Body

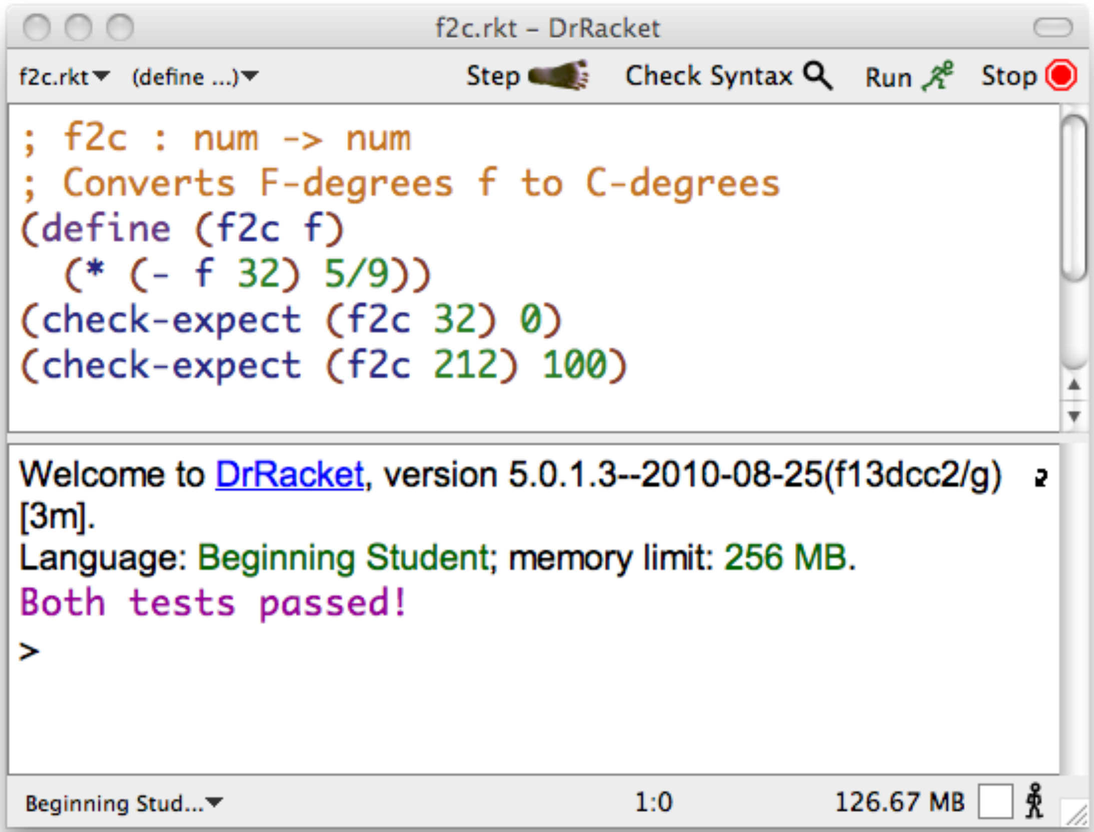
- The most creative step: implement the function body

## Test

- Run the examples

## Test

Click **Run** — examples serve as tests



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

```
Welcome to DrRacket, version 5.0.1.3--2010-08-25(f13dcc2/g)
[3m].
Language: Beginning Student; memory limit: 256 MB.
Both tests passed!
>
```

## Design Recipe - Each Step Has a Purpose

**Data**

- Shape of input data will drive the implementation

**Signature, 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**

# 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**
- 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.