Computing versus Programming

Computing

(* (– 212 32) 5/9)

→ (* 180 5/9)

→ 100

Programming

Convert °F to °C...

(define (f2c f)
  (* (– f 32) 5/9))
How to Design Programs

Programming always requires creativity

But a design recipe can guide and focus creativity

We’ll start with a simple recipe

Later, we’ll expand the recipe
Design Recipe I

**Data**

- Understand the input data: **num, bool, string, 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 → string
- Faces → image
- Wages → num
- ...

In definitions: none for now
Contract, Purpose, and Header

**Contract**

Describes input(s) and output data

- \( f2c : \text{num} \rightarrow \text{num} \)
- \( \text{is-milk?} : \text{string} \rightarrow \text{bool} \)
- \( \text{wearing-glasses?} : \text{image image image image} \rightarrow \text{bool} \)
- \( \text{netpay} : \text{num} \rightarrow \text{num} \)

**In definitions:** a comment

\[
; f2c : \text{num} \rightarrow \text{num}
\]
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 string for milk
- Checks whether \(p2\) is \(p1\) wearing glasses \(g\)
- Computes net pay (less taxes) for \(n\) hours worked

*In definitions: 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 mentioned 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

**In definitions:** as above, but absorbed into implementation

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

In definitions: 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)
Fill in the body under the header

\[
\text{(define } (f2c \ f) \\
\quad (* (- \ f \ 32) \ 5/9))
\]

\[
\text{(define } (is-milk? \ s) \\
\quad (string=? \ s \ "milk")\)
\]

**In definitions:** complete at this point

; \ f2c \ : \ num \ \rightarrow \ num
; \ Convert \ F-degrees \ f \ to \ C-degrees
\text{(define } (f2c \ f) \\
\quad (* (- \ f \ 32) \ 5/9))
\text{(check-expect } (f2c \ 32) \ 0)
\text{(check-expect } (f2c \ 212) \ 100)\)
Test

Click **Run** — 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
The Design Recipe

Use it for small tasks

so that you’ll know how to use it for BIG tasks