Symbolic Information (Ch. 5)

- Non-numeric information.
- Scheme supports three kinds of symbolic information:
  - symbols `'hello 'red`
  - strings `"Hello World"`
  - images

Symbols

- Start with single quote
  - first character must not be a digit.
- No inherent meaning to the computer!
  - functions make use of specific symbols
  - often many functions working together
- Symbols are *atomic*

Symbol Usage Examples

- You could have functions that deal with navigation instructions:

```
'East   'West   'North   'South
```

- Other possibilities: colors, foods, options, attribute names.
- In general: anything from a *discrete* domain with fixed names for elements.
- Symbols are not manipulated (changed)!

Basic Symbols Operations

- There is only one!

`symbol=?`

evaluates to `true` or `false`

- Examples:

```
(symbol=? 'hi 'fred)
```

```
(symbol=? 'hi x)
```

```
(symbol=? 'hi (somefunc blah))
```

`symbol=?` issues

- case sensitive:

  `(symbol=? 'Fred 'fred)` is `false`

- don't forget the quote!

  ```
  (symbol=? fred 'fred)
  ```

- variable names vs. symbols

  ```
  (symbol=? x 'fred)
  ```

Example function that replies

```
;; reply : symbol  ->  symbol
;; to determine a reply for the greeting s
(define (reply s) ...)
```

- For example, we might want this:

```
(reply 'GoodMorning)
'WhatIsGoodAboutIt?
```

Example `reply` function

```
(define (reply s)
  (cond
    [(symbol=? s 'GoodMorning) 'Hi]
    [(symbol=? s 'HowAreYou?) 'Fine]
    [(symbol=? s 'GoodAfternoon) 'INeedANap]
    [(symbol=? s 'GoodEvening) 'BoyAmITired]))
```

Exercise 5.1.2

- Develop the function check-guess. It consumes two numbers, `guess` and `target`. Depending on how `guess` relates to `target`, the function produces one of the following three answers: `'TooSmall`, `'Perfect`, or `'TooLarge`.

- This function will be part of a number guessing game (the rest of the code is provided in the teachpack `guess.ss`)

check-guess

```
;; check-guess number number -> symbol
;; compares guess to target and evaluates to:
;;  if guess < target : 'TooSmall
;;  if guess > target : 'TooLarge
;;  if guess = target : 'Perfect
(define (check-guess guess target) ...)
```

Testing `check-guess`

```
;; test function for check-guess
(define (test-check-guess guess target expected)
   (symbol=?
      • (check-guess guess target)
      • expected)))

(test-check-guess 10 20 'TooSmall)
(test-check-guess 30 20 'TooLarge)
(test-check-guess 10 10 'Perfect)
```

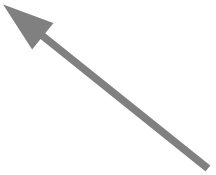
Writing `check-guess`

- We can create `check-guess` with a single `cond` expression, the basic form could look like this:

```
(define (check-guess guess target)
    (cond
      [ (< guess target) ...]
      [ (> guess target) ...]
      [ (= guess target) ...]
     ))
```

Sample `check-guess`

```
(define (check-guess guess target)
    (cond
      [ (< guess target) 'TooSmall]
      [ (> guess target) 'TooLarge]
      [ (= guess target) 'Perfect]
     ))
```



Could also be done as:
`[ else 'Perfect ]`

Using the teachpack

- include the teachpack guess.ss
- run the function:

```
(guess-with-gui check-guess)
```

Exercise for you: 5.1.5

- Write down what the function consumes and produces.
- provide a description and example
- write some test code
- design the general form
- fill in the details.
- check with test code
- Use with teachpack master.ss