

CS 135 — L12: String

Luke Lu • 2025-11-11

Characters

Characters are like `char` in Java, with format `#\ character`

Examples:

`#\a` `#\α` `#\日` `#\?`

Operations

For CS135 we will only need three operations on characters (unless we indicate otherwise on an assignment or exam): `char?`, `char=?` and `char<?`

The following expressions are true: `(char? #\a)` `(char=? #\日 #\日)` `(char<? #\a #\日)`

Generally, characters with a natural “alphabetical order” are assigned numbers that reflect that order. The following expressions are true:

`(char<? #\a #\b)` `(char<? #\α #\β)` `(char<? #\A #\Z)`

But things don’t always work the way you might expect:

`(char<? #\a #\B) ⇒ false`

Strings

Strings are like `str` in Python

Some String Operations

`:: convert a string to a list of characters`

`:: string->list: Str -> (listof Char)`

`:: convert a list of characters to a string`

`:: list->string: (listof Char) -> Str`

Testing for Equality

Remember they are tested with lexicographical order

```
;; Are two lists of characters the same?
;; char-list=: (listof Char) (listof Char) -> Bool
(define (char-list=? loc0 loc1)
  (cond [(empty? loc0) (empty? loc1)]
        [(empty? loc1) false]
        [else (and (char=? (first loc0) (first loc1))
                    (char-list=? (rest loc0) (rest loc1)))]))

;; Are two strings equal?
;; str=: Str Str -> Bool
(define (str=? s0 s1)
  (char-list=? (string->list s0) (string->list s1)))

(check-expect (str=? "hello" "hello") true)
(check-expect (str=? "hello" "ciao") false)
(check-expect (str=? "" "") true)
```

💡 **Tip** — Many string questions can be solved with string->list then do operation then list->string

Dictionaries

lookup

```
;; Lookup a key-value pair in an association list
;; lookup: Str AL -> (anyof Num empty)
(define (lookup key al)
  (cond [(empty? al) empty]
        [(string=? (first (first al)) key) (second (first al))]
        [else (lookup key (rest al))]))
```

```
(check-expect (lookup "bjones" marks) 64)
(check-expect (lookup "dleted" marks) empty)
```

delete

```
;; Delete a key-value pair from an association list
;; delete: Str AL -> AL
(define (delete key al)
  (cond [(empty? al) empty]
        [(string=? (first (first al)) key) (rest al)]
        [else (cons (first al) (delete key (rest al)))]))

(check-expect(delete "bjones" marks) (list (list "asmith" 87) (list "cwang" 87)))
(check-expect (delete "a37858w" empty) empty)
```

add

The add operator

```
;; Add a key-value pair to an association list
;; add: Str Num al -> al
(define (add key value al)
  (cons (list key value) (delete key al)))
```