[[Programmazione e laboratorio-info]]

[[Laboratorio-Scheme]]

Scheme λ

> [!info]- LINKs - https://gustavus.edu/mcs/max/concrete-abstractions-pdfs/index.html - *Concrete Abstractions* book - https://gustavus.edu/mcs/max/concabs/code/ - code for the book
>
> - Lezioni
> - https://it.wikipedia.org/wiki/Scheme

Astrazione Procedurale

> Consiste nel descrivere tutti i sottoproblemi in cui un problema è descrivibile sostituendo a queste descrizioni una chiamata ad un sottoprogramma, il cui compito sarà quello di risolvere il corrispondente sottoproblema.

[[B473D7B8-37C2-413B-BBC4-3112E5076D99.png|300]] Language: **Intermediate Studente with Lambda** Codice per calcolare la **superficie di un cilindro**: [[Scheme-superficie-cilindro|file]] Codice per declinare al **plurale i sostantivi regolari**: [[Espressione che declini al plurale i sostantivi della lingua italiana|file]]

Procedura per verificare se una parola è maschile o femminile: [[Procedura per verificare se parola è maschile o femminile|codice]]

Procedura per passare da infinito a participio passato [[Da infinito a participio passato|codice]]

Programma per la [[Generalizzazione del plurale]]

[[Tipi numerici]]: In Scheme i numeri possono essere Interi, Razionali, Double, Complessi, Costanti, Misti:

2022-10-17 12:41

Complemento a uno (invertire stringa binaria in modo ricorsivo)

```
;scheme
(define
com-
pluno ;
val:
stringa
di 0/1
(lambda
(seq) ;
seq:
stringa
di 0/1
(if (>
(string-
length
seq) 1)
(string-
append
(bit-
compl
(sub-
string
seq 0
1))
(com-
pluno
(sub-
string
seq 1))
) (bit-
compl
seq) ;
se-
quenza
di 1 bit
)))
```

(define
bit-
compl
(lambda
(bit) ;
bit:
“0”, “1”
(if
(string=?
bit “0”)
“1” “0”
))) “‘

— 12:51
—

“‘scheme

(define
s
(lambda
(k) (if
(>= k 2)
(/ (s (-
k 2)) 2)
(if (= k
0) s0
s1))))

(define s0 (expt 2 1/4)) ; radice quarta di 2
(define s1 (expt 2 -1/4)); inversa della radice quarta di 2
“‘ # Ricorsione > Procedure che richiamano loro stesse

#todo ## 2022-10-21 8:30

[[50DED43C-55D1-4D7C-B703-7955B62D2BD8.jpeg|codice 20221021]]

2022-10-24 11:46

[[8834A672-566D-4177-B76A-945BEA80D7DD.jpeg|350]] Sotto la versione equivalente senza `let`

2022-11-04 08:37

Numeri di Stirling del secondo tipo

Nessun cioccolatino fuori e nessun piatto vuoto. I piatti sono anonimi. esempio: 4 piatti, 16 cioccolatini univoci. esempio2: 2 piatti e 3 cioccolatini. -> 3 modi possibili Esempio3: 3 piatti, 4 cioccolatini -> ' '

```
(define st
    (lambda (n k)
        (if (or (= k 1) (= k n))
        1
        (+ (st (- n 1) (- k 1)) (* k (st (- n 1) k)))
        )
    )
)
```

2022-11-07

LCS: Longest Common Subsequence

LLCS = Lenght Longest Common Subsequence

```
;; llcs(ax, by) --> k
;; llcs (ax, ay) = 1 + llcs (x, y)
;; llcs (ax, by) = max(llcs (ax, y), llcs(x, by)) se a != b
;;llcs ("", y) = llcs (x, "") = 0
```

```
(define llcs ; val: intero
  (lambda(u v) ;u, v: stringhe
      (cond ((or (string=? u "") (string=? v ""))
      0)
      ((char=? (string-ref u 0))
      )
      )
  ))
```

[[F4A08797-7A9A-40F4-A918-3F25CC1AE46B.jpeg|600]]

`(llcs "ATAG""ATAG")` => 4

2022-11-14

Ricorsione ambigua, conigli e fibonacci

> [!hint]- Ipotesi L'ambiente è chiuso (situazione sperimentale!): All'istante iniziale t = 0 c'è una coppia di conigli fertile; Una coppia di conigli fertile all'istante t da' alla luce una nuova coppia di conigli ad ogni mese successivo 141, 142, I conigli nati all'istante t diventano fertili esattamente dopo un mese, all'istante 1+1; I conigli non muoiono nell'intervallo di tempo considerato; I conigli nascono sempre a coppie: un maschio e una femmina

- t = 0 -> 1 coppia fertile
- t = 1 -> 1 coppia fertile + 1 coppia
- t = 2 -> 2 coppie fertile + 1 coppia

t: *f* coppie fertili + *c* coppie t+1: *f*+*c* coppie fertili + *f* coppie cucciole Formule ricorsive: $cf(t+1) = cf(t) + c(t)\, c(t+1) = cf(t)$

```
(define coppie   ; val: intero
  (lambda (t); t: intero non negativo (clock)
    (if (= t 0)
      (coppie-fertili (- t 1))
    )
  ))

(define coppie-fertili
  (lambda (t)
    (if (= t 0)
      (+ (coppie-fertili (- t 1)) (coppie (- t 1)))
    )

  ))

(define coppie
  (lambda (t)
      (+ (coppie-fertili t) (coppie t))
  ))
```

`(coppie 0)` => 1 `(coppie 1)` => 2 `(coppie 2)` => 3 `(coppie 3)` => 5 `(coppie 12)` => 377
^ Sequenza di Fibonacci

2022-11-18

```
(define mul
  (lambda (m n)  ; n,m: interi non negativi.
    (cond ((= n 0)
           0)
          ((even? n)
           (mul (* 2 m) (quotient n 2)))
          (else
           (+ m (mul (* 2 m) (quotient n 2)))
```

```
            )
            )
      ))
```

```
(define mcd
  (lambda (x y)
    (cond ((= x y)
           x)
          ((< x y)
           (mcd x (- y x)))
          (else
           (mcd (- x y) y))
          )
          ))
```

2022-11-21

Correttezza (?)

```
(define unknown
  (lambda (x)
  (if (= x 0)
      0
      (+ (unknown (- x 1)) (odd x))
      ))
)


(define odd
  (lambda (i)
  (if (= i 1)
  1
  (+ (odd (- i 1)) 2)
)
    ))
```

Dimostrazione per [[Principio-Induzione|induzione]]

Procedure con argomenti e/o valori procedurali

1. Procedure con argomenti procedurali

2. Procedure con valori procedurali
3. Procedure con argomenti e valori procedurali

Regola di criptazione: ad ogni lettera viene applicata la stessa regola di codifica indipendentemente dal contesto. #### Cesare > La regola di Giulio Cesare per la criptazione consiste nel sostituire ogni lettera del testo originale con la lettera che si trova un certo numero di posizioni più avanti nell'alfabeto. Ad esempio, se si sceglie uno spostamento di tre posizioni, la lettera "a" diventerà "d", la "b" diventerà "e" e così via. Questo metodo di criptazione è stato utilizzato da Giulio Cesare per inviare messaggi segreti ai suoi generali durante le guerre galliche.

> [!done]+ Procedura con *argomenti* procedurali
>
> ```
> (define encryption
>   (lambda (message rule)  ; rule è la procedura
>       (if (= (string-length message)0)
>       ""
>       (string-append
>           (string (rule (string-ref message 0)))
>           (encryption (substring message 1) rule)
>           )
>       )
>   ))
> ```
>
> `(encryption "PROGRAMMAZIONE"(lambda (x)x))` cosa ritorna?

> [!done]+ Procedura con *valori* procedurali
>
> ```
> (define caesar-cipher ; valore procedurale: lettera -> lettera
>   (lambda (rot)           ; la funzione dipende da rot: integer
>       (lambda letter)
>           (let ((c (+ (char->integer letter) rot)))
>               (if (> c codZ)
>                   (integer -> char (- c 26)) ; 26 lettere dell'
>                      alfabeto
>                   (integer -> char c)
>                   ))
>           )
>   ))
>   (define codA (char -> integer #\A)) ; Costante con codice ASCII
>      per la lettera A
>   (define codZ (char -> integer #\Z)) ; Costante con codice ASCII
>      per la lettera Z
> ```

> [!attention]- Esercizio Che valore assume la seguente espressione? Perchè? `(encryption (encryption "PROGRAMMAZIONE"(caesar-cipher 3))(caesar-cipher 23))`

> >[!tip]- Soluzione > #todo

Esercizi Esame

[[20220129-Esame-Scheme]]