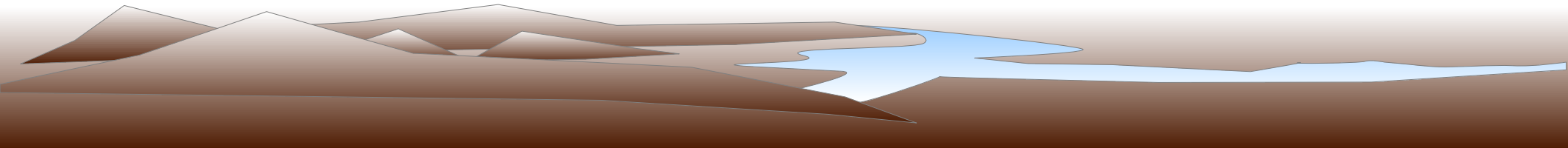
Master in Bioinformatica

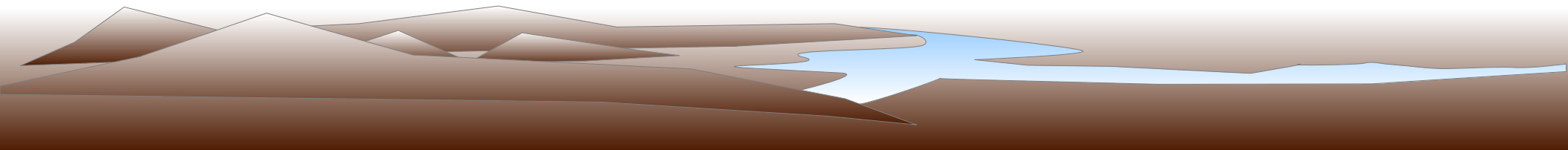
Programmazione in Perl
Introduzione alle funzioni

Andrea Sterbini
sterbini@di.uniroma1.it



Cos'è una funzione

- E' una parte di codice riusabile che normalmente:
  - Riceve dei valori da elaborare (gli “argomenti”)
  - Può manipolare variabili globali (sconsigliato)
  - Rende un valore (il “risultato”) a chi la chiama
- A che serve
  - Per riusarla in più punti del programma o in altri programmi
  - Per rendere più chiaro e leggibile il codice
  - Per risolvere alcuni problemi ricorsivi

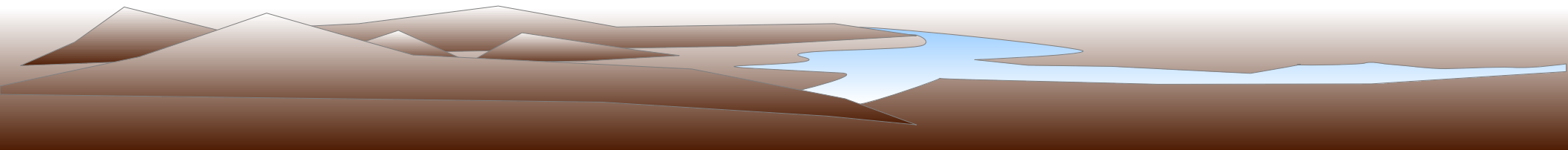


Come si esegue una funzione

- Per passare dati usate gli argomenti della funzione

```
F($arg1, $arg2, $arg3);
```

  - Vengono passati al codice della funzione nella variabile speciale `@_`
- EVITATE di modificare variabili globali
  - È più chiaro usare gli argomenti della funzione per indicare quali informazioni le sono fornite
  - Usando le variabili globali potete creare effetti collaterali che danno bugs difficili da scoprire



Come si definisce una funzione

- Si definisce il nome ed il blocco di codice

```
sub nomefunzione {
    # estraggo gli argomenti
    my ($arg1, $arg2, $arg3) = @_ ;

    # codice del corpo della funzione

    return $risultato;
}
```

- Se il numero di argomenti è fisso conviene scrivere

```
sub nomefunzione ($$$) { # segue ...
```

Strano: numero di argomenti variabile

- Il numero di argomenti può essere variabile
  - ma ricordate che le liste vengono “schiacciate”
  - quindi passate UNA SOLA LISTA **PER ULTIMA**

```
# chiamata della funzione
nomefunzione( 1, 2, 3, (4, 5, 6) );


# definizione della funzione
sub nomefunzione {
    # estraggo gli argomenti
    my ($arg1, $arg2, $arg3, @lista) = @_ ;
```

Che strano: tornare più risultati

- Il numero di valori risultanti può essere variabile
  - basta tornare una lista di valori ...

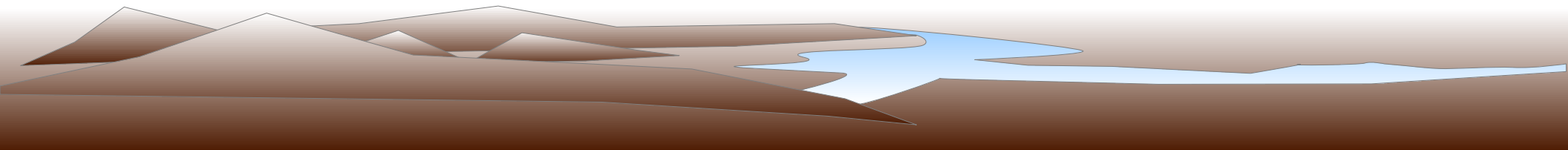
```
sub F {
  # codice della funzione ...
  return ($uno, $due, $tre);
  }
```

  - ... ed usare un assegnamento multiplo per separarli

```
my ($a, $b, $c) = F();
```

  - ... possiamo tornare UNA LISTA PER ULTIMA

```
my ($a, $b, $c, @d ) = F();
```



Funzioni ricorsive

- Una funzione può chiamare sé stessa
  - Esempio: la funzione “fattoriale(N)” definita come segue:
    - F(N) = 1 se N=0
    - F(N) = N*F(N-1) altrimenti

```
sub fattoriale ($) {
    my ($N) = @_;   # leggo l'argomento N
    if ($N == 0) {
      return 1;
    } else {
      return $N * fattoriale($N-1);
    }
  }
```

Digressione: come costruire strutture

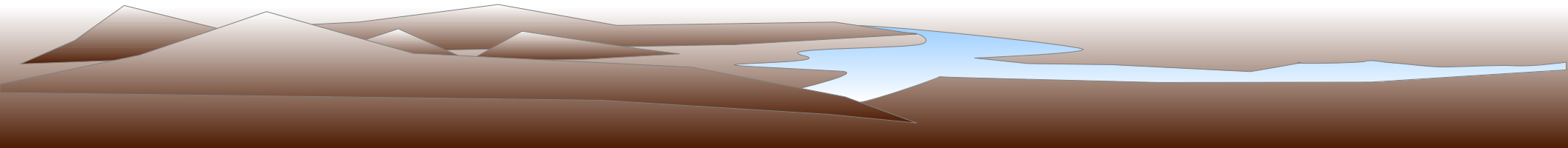
- **Problema**: liste automaticamente "schiacciate"

  ```
  ( 1, 2, 3, (4, 5, (6, 7), 8 ) )
  ```

  diventa

  ```
  ( 1, 2, 3, 4, 5, 6, 7, 8)
  ```

- **Soluzione**: usare i "**riferimenti**" che trasformano una struttura dati (lista o hash-table) in un valore **scalare**
  - Possono essere usati come valori di una lista/hash-table
    - Simili ai puntatori C (ma sicuri)
    - Simili ai riferimenti Java



Come usare i riferimenti

- Si **definiscono** usando l'operatore "\" (backslash)

```
my @lista = (1, 2, 3, 4);
my $riferimento = \$lista;
```

- Si **usano** con l'operatore "**$**"

```
my @lista2 = $$riferimento;
```

- Oppure usando **@{ }** o **%{ }**

```
my @lista2 = @{$riferimento};
my @lista2 = @$riferimento;
```