Εισαγωγή στη Haskell

Γιάννης Κασσιός

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Δομή Σημερινής Διάλεξης

- Πρόσβαση στη Haskell / Χρήση Διερμηνέα
- Εκφράσεις / Προτεραιότητα / Προσεταιριστικότητα
- Βασικοί Τύποι
- Προγράμματα
  - σχόλια, προτάσεις, καθορισμοί τύπων, ορισμοί
- Συναρτήσεις
  - τύποι και ορισμοί
  - currying/σύνθεση
  - τελεστές και συναρτήσεις
  - φρουροί
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα
- Επισκόπηση

Πρόσβαση στη Haskell

- Hugs98 (διερμηνέας Haskell):
  - Download: `http://haskel.org/hugs/`
  - Μηχανήματα Linux: `home/appl/hugs98/bin/hugs`
- Εναλλακτικός διερμηνέας/μεταγλωττιστής: Glaskow Haskell Compiler (GHC):
  - `http://haskel.org/ghc`

Χρήση Διερμηνέα

- Εμφανίζει κονσόλα εισόδου
- Ο χρήστης μπορεί να γράψει εντολές ή εκφράσεις προς αποτίμηση
- Οι εντολές αρχίζουν με το χαρακτήρα `:`
  - `:load`
  - `:cd`
  - `:reload`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Εκφράσεις Haskell

Μία έκφραση Haskell είναι:

- Μία σταθερά
- Ένα αναγνωριστικό που αρχίζει με πεζό γράμμα
- Εφαρμογή τελεστή σε άλλες εκφράσεις, πχ. `1+2`
- Εφαρμογή συνάρτησης σε άλλες εκφράσεις, πχ. `f x y`
- Μία άλλη έκφραση σε παρένθεση

Παράδειγμα έκφρασης: `(-b+sqrt(a^2-4*a*c))/(2*a*c)`

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Προτεραιότητα / Προσεταιριστικότητα

- Αποτίμηση της `1+2*3` δίνει `7` και όχι `9`
  - ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον `+`
  - παράκαμψη προτεραιότητας με παρενθέσεις: `(1+2)*3`
- Αποτίμηση της `1-1-1` δίνει `-1`
  - ο `-` έχει αριστερή προσεταιριστικότητα: `1-1-1=(1-1)-1=-1`
  - ο `^` έχει δεξιά προσεταιριστικότητα: `2^2^3 = 2^(2^3) = 2^8 = 256`
  - υπάρχουν τελεστές χωρίς προσεταιριστικότητα: η Haskell απαγορεύει: `x==y==z`
- Πίνακας Προτεραιότητας/Προσεταιριστικότητας στις Σημειώσεις

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Bool`

- `Bool`: αληθοτιμές
- Σταθερές: `True` `False`
- Τελεστές:
  - σύζευξη (λογικό "και"): `&&`
  - διάζευξη (λογικό "ή"): `||`
  - ισότητα ("ισοδυναμία"): `==`
  - ανισότητα (αποκλειστικό "ή"): `/=`
  - άρνηση: `not`
- Ισότητα και ανισότητα υποστηρίζονται για κάθε βασικό τύπο
  - η ισότητα γράφεται ==
  - ο τελεστής = είναι δεσμευμένος για ορισμούς

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Bool`

- `Bool`: αληθοτιμές
- Σταθερές: `True` `False`
- Τελεστές:
  - σύζευξη (λογικό "και"): `&&`
  - διάζευξη (λογικό "ή"): `||`
  - ισότητα ("ισοδυναμία"): `==`
  - ανισότητα (αποκλειστικό "ή"): `/=`
  - άρνηση: `not`
- Ισότητα και ανισότητα υποστηρίζονται για κάθε βασικό τύπο
  - η ισότητα γράφεται ==
  - ο τελεστής = είναι δεσμευμένος για ορισμούς

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Bool`

- `Bool`: αληθοτιμές
- Σταθερές: `True` `False`
- Τελεστές:
  - σύζευξη (λογικό "και"): `&&`
  - διάζευξη (λογικό "ή"): `||`
  - ισότητα ("ισοδυναμία"): `==`
  - ανισότητα (αποκλειστικό "ή"): `/=`
  - άρνηση: `not`
- Ισότητα και ανισότητα υποστηρίζονται για κάθε βασικό τύπο
  - η ισότητα γράφεται `==`
  - ο τελεστής `=` είναι δεσμευμένος για ορισμούς

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Bool`

- `Bool`: αληθοτιμές
- Σταθερές: `True` `False`
- Τελεστές:
  - σύζευξη (λογικό "και"): `&&`
  - διάζευξη (λογικό "ή"): `||`
  - ισότητα ("ισοδυναμία"): `==`
  - ανισότητα (αποκλειστικό "ή"): `/=`
  - άρνηση: `not`
- Ισότητα και ανισότητα υποστηρίζονται για κάθε βασικό τύπο
  - η ισότητα γράφεται ==
  - ο τελεστής = είναι δεσμευμένος για ορισμούς

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Int`

- `Int`: ακέραιοι αριθμοί
- Σταθερές: `0 3 10 -7` κτλ.
- Τελεστές: `+ - * ^`
- Συναρτήσεις: `div mod abs`
- Τελεστές σύγκρισης: `== /= < <= => >`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Int`

- `Int`: ακέραιοι αριθμοί
- Σταθερές: `0 3 10 -7` κτλ.
- Τελεστές: `+ - * ^`
- Συναρτήσεις: `div mod abs`
- Τελεστές σύγκρισης: `== /= < <= => >`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Int`

- `Int`: ακέραιοι αριθμοί
- Σταθερές: `0 3 10 -7` κτλ.
- Τελεστές: `+ - * ^`
- Συναρτήσεις: `div mod abs`
- Τελεστές σύγκρισης: `== /= < <= => >`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Int`

- `Int`: ακέραιοι αριθμοί
- Σταθερές: `0 3 10 -7` κτλ.
- Τελεστές: `+ - * ^`
- Συναρτήσεις: `div mod abs`
- Τελεστές σύγκρισης: `== /= < <= => >`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Int`

- `Int`: ακέραιοι αριθμοί
- Σταθερές: `0 3 10 -7` κτλ.
- Τελεστές: `+ - * ^`
- Συναρτήσεις: `div mod abs`
- Τελεστές σύγκρισης: `== /= < <= => >`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Char`

- `Char`: χαρακτήρες
- Σταθερές: `'a'` `'C'` `'7'` κτλ.
- Επίσης: `'\t'` `'\n'` `'\\'` `'\''` `'\"'`
- Συναρτήσεις: `chr` `ord`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Char`

- `Char`: χαρακτήρες
- Σταθερές: `'a'` `'C'` `'7'` κτλ.
- Επίσης: `'\t'` `'\n'` `'\\'` `'\''` `'\"'`
- Συναρτήσεις: `chr` `ord`

Βασικοί Τύποι: `Bool, Int, Char, Float`

`Char`

- `Char`: χαρακτήρες
- Σταθερές: `'a' 'C' '7'` κτλ.
- Επίσης: `'\t' '\n' '\\' '\'' '\"'`
- Συναρτήσεις: `chr ord`

Βασικοί Τύποι: `Bool, Int, Char, Float`

`Char`

- `Char`: χαρακτήρες
- Σταθερές: `'a' 'C' '7'` κτλ.
- Επίσης: `'\t' '\n' '\\' '\'' '\"'`
- Συναρτήσεις: `chr ord`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:

```
fromIntegral(floor pi) + 2.5
```

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:

```
fromIntegral(floor pi) + 2.5
```

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:

```
fromIntegral(floor pi) + 2.5
```

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:

```
fromIntegral(floor pi) + 2.5
```

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:

```
fromIntegral(floor pi) + 2.5
```

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:
  `fromIntegral(floor pi) + 2.5`

Βασικοί Τύποι: `Bool`, `Int`, `Char`, `Float`

`Float`

- `Float`: αριθμοί κινητής υποδιαστολής
- Σταθερές: `2.0 -1.4 2e-7 pi`
- Τελεστές: `+ - * / ** ^`
- Συναρτήσεις: `abs ceiling floor round cos sin tan log sqrt`
- Μετατροπή ακεραίου: `fromIntegral`
- Αυτόματες μετατροπές δε γίνονται:
  `fromIntegral(floor pi) + 2.5`

Προγράμματα Haskell

- Πρόγραμμα Haskell (Haskell script): αρχείο κειμένου με κατάληξη `.hs`
- Σύνολο ορισμών ονομάτων
- Φορτώνεται με την εντολή `:load`
- Παράδειγμα: πρόγραμμα `firsthaskell.hs`

```
one::Int
one = 1
two::Int
two = 2
```

Προγράμματα Haskell

- Πρόγραμμα Haskell (Haskell script): αρχείο κειμένου με κατάληξη `.hs`
- Σύνολο ορισμών ονομάτων
- Φορτώνεται με την εντολή `:load`
- Παράδειγμα: πρόγραμμα `firsthaskell.hs`

```
one::Int
one = 1
two::Int
two = 2
```

Προγράμματα Haskell

- Πρόγραμμα Haskell (Haskell script): αρχείο κειμένου με κατάληξη `.hs`
- Σύνολο ορισμών ονομάτων
- Φορτώνεται με την εντολή `:load`
- Παράδειγμα: πρόγραμμα `firsthaskell.hs`

```
one::Int
one = 1
two::Int
two = 2
```

Προγράμματα Haskell

- Πρόγραμμα Haskell (Haskell script): αρχείο κειμένου με κατάληξη `.hs`
- Σύνολο ορισμών ονομάτων
- Φορτώνεται με την εντολή `:load`
- Παράδειγμα: πρόγραμμα `firsthaskell.hs`

```
one::Int
one = 1
two::Int
two = 2
```

- Βρίσκουμε το πρόγραμμα με την εντολή
  `:cd /path/to/script`
- Φορτώνουμε τους ορισμούς του προγράμματος
  `:load firsthaskell`
- Οι ορισμοί του προγράμματος είναι διαθέσιμοι στο διερμηνέα
  - η έκφραση `one+two` αποτιμάται σε `3`
- Σε περίπτωση αλλαγής του προγράμματος, το ξαναφορτώνουμε
  `:reload`

- Βρίσκουμε το πρόγραμμα με την εντολή
  `:cd /path/to/script`
- Φορτώνουμε τους ορισμούς του προγράμματος
  `:load firsthaskell`
- Οι ορισμοί του προγράμματος είναι διαθέσιμοι στο διερμηνέα
  - η έκφραση `one+two` αποτιμάται σε 3
- Σε περίπτωση αλλαγής του προγράμματος, το ξαναφορτώνουμε
  `:reload`

- Βρίσκουμε το πρόγραμμα με την εντολή
  `:cd /path/to/script`
- Φορτώνουμε τους ορισμούς του προγράμματος
  `:load firsthaskell`
- Οι ορισμοί του προγράμματος είναι διαθέσιμοι στο διερμηνέα
  - η έκφραση `one+two` αποτιμάται σε `3`
- Σε περίπτωση αλλαγής του προγράμματος, το ξαναφορτώνουμε
  `:reload`

- Βρίσκουμε το πρόγραμμα με την εντολή
  `:cd /path/to/script`
- Φορτώνουμε τους ορισμούς του προγράμματος
  `:load firsthaskell`
- Οι ορισμοί του προγράμματος είναι διαθέσιμοι στο διερμηνέα
  - η έκφραση `one+two` αποτιμάται σε `3`
- Σε περίπτωση αλλαγής του προγράμματος, το ξαναφορτώνουμε
  `:reload`

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με `--` και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου `;` επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

- Ένα πρόγραμμα απότελείται από σχόλια(comments) και προτάσεις (sentences)
- Τα σχόλια αρχίζουν οπουδήποτε μέσα στο πρόγραμμα με -- και τελειώνουν στο τέλος γραμμής
  - αγνοούνται από το διερμηνέα
- Μία πρόταση
  - μπορεί να αρχίζει σε οποιαδήποτε στήλη
  - καταλαμβάνει και όλες τις γραμμές που αρχίζουν δεξιότερα από αυτήν
  - η πρώτη γραμμή που δεν αρχίζει δεξιότερα από αυτήν είναι η πρώτη γραμμή της επόμενης πρότασης
  - off-side rule
- Η χρήση του συμβόλου ; επίσης διαχωρίζει προτάσεις

Παράδειγμα: Τα παρακάτω αρχεία περιέχουν ισοδύναμες προτάσεις:

```
one = 0+1
two = 2
```

```
one --comment
 =
  0+ --comment
 1
two = 2
```

```
 one
  =
   0+
  1
two = 2
```

```
one = 0+1 ; two = 2
```

Μία πρόταση είναι ένα από τα παρακάτω:

- Καθορισμός τύπου (type annotation) ενός αναγνωριστικού:
  αναγνωριστικό : : *τύπος*
  (όχι απαραίτητο, αλλά συνηθίζεται)
- Ορισμός (definition) ενός αναγνωριστικού
  αναγνωριστικό=έκφραση
- Στη συνέχεια θα ασχοληθούμε με ορισμούς συναρτήσεων

Μία πρόταση είναι ένα από τα παρακάτω:

- Καθορισμός τύπου (type annotation) ενός αναγνωριστικού:
  αναγνωριστικό : : *τύπος*
  (όχι απαραίτητο, αλλά συνηθίζεται)
- Ορισμός (definition) ενός αναγνωριστικού
  αναγνωριστικό=έκφραση
- Στη συνέχεια θα ασχοληθούμε με ορισμούς συναρτήσεων

Μία πρόταση είναι ένα από τα παρακάτω:

- Καθορισμός τύπου (type annotation) ενός αναγνωριστικού:
  αναγνωριστικό : : *τύπος*
  (όχι απαραίτητο, αλλά συνηθίζεται)
- Ορισμός (definition) ενός αναγνωριστικού
  αναγνωριστικό=έκφραση
- Στη συνέχεια θα ασχοληθούμε με ορισμούς συναρτήσεων

- Τύπος συνάρτησης με παραμέτρους τύπου $A, B, C, \ldots$ και αποτέλεσμα επιστροφής R:
  $A\text{->}B\text{->}C\text{->}...\text{->}R$
- Παράδειγμα:
  `add::Int->Int->Int`
- Ορισμός συνάρτησης: πριν το = δίνουμε όνομα στις παραμέτρους
  - τυπικές παράμετροι (formal parameters)
  - είναι ορατές μόνο στον τρέχοντα ορισμό
- Παράδειγμα:
  `add x y = x+y`

- Τύπος συνάρτησης με παραμέτρους τύπου $A, B, C, \ldots$ και αποτέλεσμα επιστροφής R:
  $A\text{->}B\text{->}C\text{->}...\text{->}R$
- Παράδειγμα:
  `add::Int->Int->Int`
- Ορισμός συνάρτησης: πριν το = δίνουμε όνομα στις παραμέτρους
  - τυπικές παράμετροι (formal parameters)
  - είναι ορατές μόνο στον τρέχοντα ορισμό
- Παράδειγμα:
  `add x y = x+y`

- Τύπος συνάρτησης με παραμέτρους τύπου $A, B, C, \ldots$ και αποτέλεσμα επιστροφής R:
  $A\text{->}B\text{->}C\text{->}\ldots\text{->}R$
- Παράδειγμα:
  `add::Int->Int->Int`
- Ορισμός συνάρτησης: πριν το = δίνουμε όνομα στις παραμέτρους
  - τυπικές παράμετροι (formal parameters)
  - είναι ορατές μόνο στον τρέχοντα ορισμό
- Παράδειγμα:
  `add x y = x+y`

- Τύπος συνάρτησης με παραμέτρους τύπου $A, B, C, \ldots$ και αποτέλεσμα επιστροφής R:
  $A\text{->}B\text{->}C\text{->}...\text{->}R$
- Παράδειγμα:
  `add::Int->Int->Int`
- Ορισμός συνάρτησης: πριν το = δίνουμε όνομα στις παραμέτρους
  - τυπικές παράμετροι (formal parameters)
  - είναι ορατές μόνο στον τρέχοντα ορισμό
- Παράδειγμα:
  ```
  add x y = x+y
  ```

- Εφαρμογή (application) της συνάρτησης: `add 2 3`
  - περνάει την τιμή `2` στη θέση της `x` και την `3` στη θέση της `y`
  - αποτίμηση της 2+3 $\rightarrow$ αποτέλεσμα `5`
  - οι εκφράσεις `2` και `3` είναι οι πραγματικές παράμετροι (actual parameters) της εφαρμογής
- Θα γράφουμε $\mathtt{E}^{\mathtt{t}}_{\mathtt{x}}$:
  - αντικατάσταση του `x` από `t` στην `E`
- Παράδειγμα: `add 2 3` = $(\mathtt{x+y})^{\mathtt{2}\ \ \mathtt{3}}_{\mathtt{x}\ \ \mathtt{y}}$ = `2+3` = `5`

- Εφαρμογή (application) της συνάρτησης: `add 2 3`
  - περνάει την τιμή `2` στη θέση της `x` και την `3` στη θέση της `y`
  - αποτίμηση της `2+3` $\rightarrow$ αποτέλεσμα `5`
  - οι εκφράσεις `2` και `3` είναι οι πραγματικές παράμετροι (actual parameters) της εφαρμογής
- Θα γράφουμε $\mathtt{E}^{\mathtt{t}}_{\mathtt{x}}$:
  - αντικατάσταση του `x` από `t` στην `E`
- Παράδειγμα: `add 2 3` $= (\mathtt{x+y})^{\mathtt{2}\ \ \mathtt{3}}_{\mathtt{x}\ \ \mathtt{y}} = \mathtt{2+3} = \mathtt{5}$

- Εφαρμογή (application) της συνάρτησης: `add 2 3`
  - περνάει την τιμή `2` στη θέση της `x` και την `3` στη θέση της `y`
  - αποτίμηση της `2+3` $\rightarrow$ αποτέλεσμα `5`
  - οι εκφράσεις `2` και `3` είναι οι πραγματικές παράμετροι (actual parameters) της εφαρμογής
- Θα γράφουμε $\mathtt{E}^{\mathtt{t}}_{\mathtt{x}}$:
  - αντικατάσταση του `x` από `t` στην `E`
- Παράδειγμα: `add 2 3` = $\mathtt{(x+y)}^{\mathtt{2}\ \mathtt{3}}_{\mathtt{x}\ \mathtt{y}}$ = `2+3` = `5`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x` = $(x+1)^{x}_{x}$ `= x+1 = 5+1 = 6`
- `f(x+1)=`$(x+1)^{x+1}_{x}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει 5
- Αποτίμηση `f 10` δίνει 11. Δεν αλλάζει την τιμή της `x`
- `f x` = $(x+1)^{x}_{x}$ = x+1 = 5+1 = 6
- `f(x+1)`=$(x+1)^{x+1}_{x}$ = (x+1)+1 = (5+1)+1=7

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x` = $(x+1)^{x}_{x}$ `= x+1 = 5+1 = 6`
- `f(x+1)=`$(x+1)^{x+1}_{x}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x = (x+1)[x/x] = x+1 = 5+1 = 6`
- `f(x+1)=(x+1)[x+1/x] = (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x` = $(x+1)^{x}_{x} = x+1 = 5+1 = 6$
- $f(x+1) = (x+1)^{x+1}_{x} = (x+1)+1 = (5+1)+1 = 7$

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x = (x+1)`$^{\texttt{x}}_{\texttt{x}}$ `= x+1 = 5+1 = 6`
- `f(x+1)=(x+1)`$^{\texttt{x+1}}_{\texttt{x}}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x = (x+1)`$^{\texttt{x}}_{\texttt{x}}$ `= x+1 = 5+1 = 6`
- `f(x+1)=(x+1)`$^{\texttt{x+1}}_{\texttt{x}}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x = (x+1)`$^{\texttt{x}}_{\texttt{x}}$ `= x+1 = 5+1 = 6`
- `f(x+1)=`$(\texttt{x+1})^{\texttt{x+1}}_{\texttt{x}}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- `f x = (x+1)`$^{\texttt{x}}_{\texttt{x}}$ `= x+1 = 5+1 = 6`
- `f(x+1)=(x+1)`$^{\texttt{x+1}}_{\texttt{x}}$ `= (x+1)+1 = (5+1)+1=7`

```
f x = x + 1
x = 5
```

- Η τυπική παράμετρος `x` δεν έχει σχέση με τον ακέραιο `x`
  - `x` τοπικά ορατή στον ορισμό της `f`
  - `x` καθολικά ορατή
- Αποτίμηση `x` δίνει `5`
- Αποτίμηση `f 10` δίνει `11`. Δεν αλλάζει την τιμή της `x`
- $\texttt{f x} = \texttt{(x+1)}^{\texttt{x}}_{\texttt{x}} = \texttt{x+1} = \texttt{5+1} = \texttt{6}$
- $\texttt{f(x+1)=(x+1)}^{\texttt{x+1}}_{\texttt{x}} = \texttt{(x+1)+1} = \texttt{(5+1)+1=7}$

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής `->` έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής `->` έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής `->` έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής `->` έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής -> έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής -> έχει δεξιά προσεταιριστικότητα

- Αν περάσουμε λιγότερες παραμέτρους σε μία συνάρτηση, τότε επιστρέφεται μία συνάρτηση που περιμένει και τις υπόλοιπες παραμέτρους
- Για παράδειγμα:

```
add::Int->Int->Int
add 2::Int->Int
add 2 3::Int
```

- Η `add 2` είναι μία συνάρτηση που αυξάνει το όρισμά της κατά 2. Πχ. `(add 2)3=5`
- Δεν υπάρχει διαφορά μεταξύ `add 2 3` και `(add 2)3`
  - η εφαρμογή έχει αριστερή προσεταιριστικότητα
- Δεν υπάρχει διαφορά μεταξύ `Int->Int->Int` και `Int->(Int->Int)`

```
add::Int->(Int->Int)
```

  - ο τελεστής -> έχει δεξιά προσεταιριστικότητα

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z` $= ((x + y)^{y}_{x})^{z}_{y} = (y+y)^{z}_{y} =$ `2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  `add x w = x+w`

  `f z = (add y)z` $= (x+w)^{y}_{x}\,{}^{z}_{w} =$ `y+z = 5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z` $= ((x + y)^{y}_{x})^{z}_{y} =$ `(y+y)z` $=$ `2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  `add x w = x+w`

  `f z = (add y)z` $= (x+w)^{y}_{x}\,{}^{z}_{w} =$ `y+z` $=$ `5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z` $= ((x + y)^{y}_{x})^{z}_{y} =$ `(y+y)z` $=$ `2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  `add x w = x+w`

  `f z = (add y)z` $= (x+w)^{y}_{x}\ ^{z}_{w} =$ `y+z` $=$ `5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z` $= ((x + y)^{y}_{x})^{z}_{y} = (y+y)^{z}_{y}$ `= 2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  `add x w = x+w`

  `f z = (add y)z` $= (x+w)^{y}_{x}\,{}^{z}_{w}$ `= y+z = 5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z` $= ((x + y)^{y}_{x})^{z}_{y} = (y+y)^{z}_{y}$ `= 2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  ```
  add x w = x+w
  ```

  `f z = (add y)z` $= (x+w)^{y}_{x}\ ^{z}_{w}$ `= y+z = 5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z = ((x + y)`$^{\texttt{y}}_{\texttt{x}}$`)`$^{\texttt{z}}_{\texttt{y}}$ `= (y+y)`$^{\texttt{z}}_{\texttt{y}}$ `= 2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  ```
  add x w = x+w
  f z = (add y)z = (x+w)^y_x ^z_w = y+z = 5+z
  ```

- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

```
add x y = x+y
y = 5
f = add y
```

- Ποια είναι η συνάρτηση `f`;
- `f z = (add y)z = ((x + y)`$^{\texttt{y}}_{\texttt{x}}$`)`$^{\texttt{z}}_{\texttt{y}}$ `= (y+y)`$^{\texttt{z}}_{\texttt{y}}$ `= 2*z` και άρα `f z = 2*z` ΛΑΘΟΣ!
- Αλλαγή ονόματος τυπικής παραμέτρου:

  ```
  add x w = x+w
  ```

  `f z = (add y)z = (x+w)`$^{\texttt{y}}_{\texttt{x}}$ $^{\texttt{z}}_{\texttt{w}}$ `= y+z = 5+z`
- Εφαρμογή συνάρτησης:
  - **πρώτα αλλάζουμε τα ονόματα τυπικών παραμέτρων** ώστε να μήν υπάρχει σύγκρουση ονομάτων
  - **μετά** κάνουμε τις αντικαταστάσεις

- Σύνθεση (composition) δύο συναρτήσεων: `f.g`
- Πρώτα εφαρμόζεται η `g`. Μετά η `f` στο αποτέλεσμα
- Ισοδύναμοι ορισμοί:

  `h=f.g`

  και

  ```
  h x = f(g x)
  ```

- Σύνθεση (composition) δύο συναρτήσεων: `f.g`
- Πρώτα εφαρμόζεται η `g`. Μετά η `f` στο αποτέλεσμα
- Ισοδύναμοι ορισμοί:

```
h=f.g
```

και

```
h x = f(g x)
```

- Σύνθεση (composition) δύο συναρτήσεων: `f.g`
- Πρώτα εφαρμόζεται η `g`. Μετά η `f` στο αποτέλεσμα
- Ισοδύναμοι ορισμοί:

  `h=f.g`

  και

  ```
  h x = f(g x)
  ```

- Προθεματική γραφή: η συνάρτηση γράφεται πριν από τα ορίσματα
  - οι συναρτήσεις της Haskell γράφονται προθεματικά
- Ενθεματική γραφή: η συνάρτηση γράφεται μεταξύ του πρώτου και του δεύτερου ορίσματος
  - οι δυαδικοί τελεστές της Haskell γράφονται ενθεματικά
- Παράδειγμα: `abs(1+x)`
- Μπορούμε να χρησιμοποιήσουμε έναν τελεστή όπως οι συναρτήσεις, με παρενθέσεις
  - `(+) 2 3`
  - `(+) 2`
  - `sum = reduce 0 (+)`

- Προθεματική γραφή: η συνάρτηση γράφεται πριν από τα ορίσματα
  - οι συναρτήσεις της Haskell γράφονται προθεματικά
- Ενθεματική γραφή: η συνάρτηση γράφεται μεταξύ του πρώτου και του δεύτερου ορίσματος
  - οι δυαδικοί τελεστές της Haskell γράφονται ενθεματικά
- Παράδειγμα: `abs(1+x)`
- Μπορούμε να χρησιμοποιήσουμε έναν τελεστή όπως οι συναρτήσεις, με παρενθέσεις
  - `(+) 2 3`
  - `(+) 2`
  - `sum = reduce 0 (+)`

- Μπορούμε να χρησιμοποιήσουμε μία συνάρτηση ενθεματικά, με αντίστροφα εισαγωγικά
  - `2 ‘add‘ 3`
- Μπορούμε να ορίσουμε δικούς μας ενθεματικούς τελεστές:

```
x ~~ y = (x+y)/2
infix 5 ~~
```

- Προσεταιριστικότητα: `infixl infixr`

- Μπορούμε να χρησιμοποιήσουμε μία συνάρτηση ενθεματικά, με αντίστροφα εισαγωγικά
  - `2 ‘add‘ 3`
- Μπορούμε να ορίσουμε δικούς μας ενθεματικούς τελεστές:

```
x ~~ y = (x+y)/2
infix 5 ~~
```

- Προσεταιριστικότητα: `infixl infixr`

- Μπορούμε να χρησιμοποιήσουμε μία συνάρτηση ενθεματικά, με αντίστροφα εισαγωγικά
  - `2 ‘add‘ 3`
- Μπορούμε να ορίσουμε δικούς μας ενθεματικούς τελεστές:

```
x ~~ y = (x+y)/2
infix 5 ~~
```

- Προσεταιριστικότητα: `infixl infixr`

- Ορισμός συνάρτησεις με "φρουρούς":
  οριζόμενη συνάρτηση και τυπικές παράμετροι
  | *συνθήκη* = *έκφραση*
  | *συνθήκη* = *έκφραση*
  ...
  | *συνθήκη* = *έκφραση*
- Πρώτη αληθής συνθήκη → αποτίμηση και επιστροφή αντίστοιχης έκφρασης
- Παράδειγμα:

```
sign::Int->Int
sign x
  | x>0  = 1
  | x==0 = 0
  | x<0  = -1
```

- Οι συνθήκες ονομάζονται φρουροί (guards)

- Ορισμός συνάρτησεις με "φρουρούς":
  οριζόμενη συνάρτηση και τυπικές παράμετροι
  | *συνθήκη* = *έκφραση*
  | *συνθήκη* = *έκφραση*
  ...
  | *συνθήκη* = *έκφραση*
- Πρώτη αληθής συνθήκη $\rightarrow$ αποτίμηση και επιστροφή αντίστοιχης έκφρασης
- Παράδειγμα:

```
sign::Int->Int
sign x
  | x>0  = 1
  | x==0 = 0
  | x<0  = -1
```

- Οι συνθήκες ονομάζονται φρουροί (guards)

- Ορισμός συνάρτησεις με "φρουρούς":
  οριζόμενη συνάρτηση και τυπικές παράμετροι
  | *συνθήκη* = *έκφραση*
  | *συνθήκη* = *έκφραση*

  ...
  | *συνθήκη* = *έκφραση*
- Πρώτη αληθής συνθήκη $\rightarrow$ αποτίμηση και επιστροφή αντίστοιχης έκφρασης
- Παράδειγμα:

```
sign::Int->Int
sign x
  | x>0  = 1
  | x==0 = 0
  | x<0  = -1
```

- Οι συνθήκες ονομάζονται φρουροί (guards)

- Ορισμός συνάρτησεις με "φρουρούς":
  οριζόμενη συνάρτηση και τυπικές παράμετροι
  | *συνθήκη* = *έκφραση*
  | *συνθήκη* = *έκφραση*
  ...
  | *συνθήκη* = *έκφραση*
- Πρώτη αληθής συνθήκη $\rightarrow$ αποτίμηση και επιστροφή αντίστοιχης έκφρασης
- Παράδειγμα:

```
sign::Int->Int
sign x
  | x>0  = 1
  | x==0 = 0
  | x<0  = -1
```

- Οι συνθήκες ονομάζονται φρουροί (guards)

Εναλλακτικοί ορισμοί της `sign`:

```
sign x
   | x>0        = 1
   | x==0       = 0
   | otherwise  = -1
```

ή

```
   | x>0  = 1
   | x==0 = 0
   | True  = -1
```

ή

```
sign x = if x>0 then 1 else if x==0 then 0 else 1
```

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  `apply::Int->Int->(Int->Int)->Int`

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  `apply::Int->Int->(Int->Int)->Int`

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  `apply::Int->Int->(Int->Int)->Int`

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  ```
  apply::Int->Int->(Int->Int)->Int
  ```

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  ```
  apply::Int->Int->(Int->Int)->Int
  ```

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  `apply::Int->Int->(Int->Int)->Int`

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:

```
apply::Int->Int->(Int->Int)->Int
```

- Ο τύπος συνάρτησης ανώτερης τάξης χρειάζεται πάντα παρενθέσεις:
  `(Int->Int)->Int` $\neq$ `Int->Int->Int`
- Παράδειγμα: επανειλημμένη εφαρμογή συνάρτησης. Ορίσματα
  - πρώτο όρισμα: αριθμός εφαρμογών `n`
  - δεύτερο όρισμα: αποτέλεσμα `i` για 0 εφαρμογές
  - τρίτο όρισμα: η συνάρτηση `f` που εφαρμόζεται πολλές φορές
- Τύπος:
  `apply::Int->Int->(Int->Int)->Int`

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
   |  n==0   = i
   |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

Ορισμός:

```
apply n i f
  |  n==0   = i
  |  n>0    = f(apply (n-1) i f)
suc x = x+1
add x y = apply y x suc
mult x y = apply y 0 (add x)
power x y = apply y 1 (mult x)
```

- Αναδρομικός ορισμός (recursive definition): χρησιμοποιούμε το υπό ορισμό όνομα:

```
factorial :: Int -> Int
factorial n
| n==0 = 1
| n>0  = n*factorial(n-1)
```

- Αμοιβαία αναδρομή (mutual recursion): ομοίως, αλλά για πολλά ονόματα:

```
e::Int->Bool
o::Int->Bool
e x = if x == 0 then True else o (abs x - 1)
o x = if x == 0 then False else e (abs x - 1)
```

- Αναδρομικός ορισμός (recursive definition): χρησιμοποιούμε το υπό ορισμό όνομα:

```
factorial :: Int -> Int
factorial n
| n==0 = 1
| n>0  = n*factorial(n-1)
```

- Αμοιβαία αναδρομή (mutual recursion): ομοίως, αλλά για πολλά ονόματα:

```
e::Int->Bool
o::Int->Bool
e x = if x == 0 then True else o (abs x - 1)
o x = if x == 0 then False else e (abs x - 1)
```

- Πρωταρχική αναδρομή (primitive recursion): το `f n` εξαρτάται μόνο από το `f(n-1)`
- Γράψτε μία συνάρτηση που υπολογίζει το τετράγωνο ενός φυσικού αριθμού, χρησιμοποιώντας μόνο πρόσθεση και αφαίρεση.

$$(n\text{-}1)^2 = n^2 - 2n + 1$$
$$n^2 = (n\text{-}1)^2 + n + n - 1$$

- Λύση στη Haskell:

```
sqr n
  | n==0 = 0
  | True = sqr(n-1) + n + n - 1
```

Παραδείγματα
Υπολογισμός Τετραγώνου με Αναδρομή

- Πρωταρχική αναδρομή (primitive recursion): το `f n` εξαρτάται μόνο από το `f(n-1)`
- Γράψτε μία συνάρτηση που υπολογίζει το τετράγωνο ενός φυσικού αριθμού, χρησιμοποιώντας μόνο πρόσθεση και αφαίρεση.

  $(n\text{-}1)^2 = n^2 \text{ - } 2n + 1$

  $n^2 = (n\text{-}1)^2 + n + n \text{ - } 1$
- Λύση στη Haskell:

```
sqr n
  | n==0 = 0
  | True = sqr(n-1) + n + n - 1
```

- Πρωταρχική αναδρομή (primitive recursion): το `f n` εξαρτάται μόνο από το `f(n-1)`
- Γράψτε μία συνάρτηση που υπολογίζει το τετράγωνο ενός φυσικού αριθμού, χρησιμοποιώντας μόνο πρόσθεση και αφαίρεση.
  $(n\text{-}1)^2 = n^2 - 2n + 1$
  $n^2 = (n\text{-}1)^2 + n + n - 1$
- Λύση στη Haskell:

```
sqr n
  | n==0 = 0
  | True = sqr(n-1) + n + n - 1
```

- Πρωταρχική αναδρομή (primitive recursion): το `f n` εξαρτάται μόνο από το `f(n-1)`
- Γράψτε μία συνάρτηση που υπολογίζει το τετράγωνο ενός φυσικού αριθμού, χρησιμοποιώντας μόνο πρόσθεση και αφαίρεση.
  $(n\text{-}1)^2 = n^2 \text{ - } 2n + 1$
  $n^2 = (n\text{-}1)^2 + n + n \text{ - } 1$
- Λύση στη Haskell:

```
sqr n
  | n==0 = 0
  | True = sqr(n-1) + n + n - 1
```

- Γράψτε μία συνάρτηση που υπολογίζει τον αριθμό των συνδυασμών που μπορούμε να έχουμε αν επιλέξουμε k αντικείμενα από n αντικείμενα.

$$\text{comb } n\ k = \frac{n!}{k!(n\text{-}k)!}$$

$$= \frac{n!}{k!(n\text{-}k)!} = \frac{(n\text{-}1)!n}{(k\text{-}1)!k((n\text{-}1)\text{-}(k\text{-}1))!} = \frac{n}{k}\text{comb}(n\text{-}1)(k\text{-}1)$$

- Λύση στη Haskell:

```
comb :: Int->Int->Int
comb n k
  | k == 0    = 1
  | True      = n * comb (n-1) (k-1) `div` k
```

- Γράψτε μία συνάρτηση που υπολογίζει τον αριθμό των συνδυασμών που μπορούμε να έχουμε αν επιλέξουμε k αντικείμενα από n αντικείμενα.

  $\text{comb } n\ k = \frac{n!}{k!(n\text{-}k)!}$

  $= \frac{n!}{k!(n\text{-}k)!} = \frac{(n\text{-}1)!n}{(k\text{-}1)!k((n\text{-}1)\text{-}(k\text{-}1))!} = \frac{n}{k}\text{comb}(n\text{-}1)(k\text{-}1)$

- Λύση στη Haskell:

```
comb :: Int->Int->Int
comb n k
  | k == 0    = 1
  | True      = n * comb (n-1) (k-1) `div` k
```

- Γράψτε μία συνάρτηση που υπολογίζει τον αριθμό των συνδυασμών που μπορούμε να έχουμε αν επιλέξουμε k αντικείμενα από n αντικείμενα.

$$\text{comb } n\ k = \frac{n!}{k!(n\text{-}k)!}$$

$$= \frac{n!}{k!(n\text{-}k)!} = \frac{(n\text{-}1)!n}{(k\text{-}1)!k((n\text{-}1)\text{-}(k\text{-}1))!} = \frac{n}{k}\text{comb}(n\text{-}1)(k\text{-}1)$$

- Λύση στη Haskell:

```
comb :: Int->Int->Int
comb n k
  | k == 0    = 1
  | True      = n * comb (n-1) (k-1) `div` k
```

- Γράψτε μία συνάρτηση που υπολογίζει τον αριθμό των συνδυασμών που μπορούμε να έχουμε αν επιλέξουμε k αντικείμενα από n αντικείμενα.

$$\text{comb } n\ k = \frac{n!}{k!(n\text{-}k)!}$$

$$= \frac{n!}{k!(n\text{-}k)!} = \frac{(n\text{-}1)!n}{(k\text{-}1)!k((n\text{-}1)\text{-}(k\text{-}1))!} = \frac{n}{k}\text{comb}(n\text{-}1)(k\text{-}1)$$

- Λύση στη Haskell:

```
comb :: Int->Int->Int
comb n k
  | k == 0    = 1
  | True      = n * comb (n-1) (k-1) `div` k
```

- Γράψτε μία συνάρτηση που υπολογίζει τον αριθμό των συνδυασμών που μπορούμε να έχουμε αν επιλέξουμε k αντικείμενα από n αντικείμενα.

$$\text{comb } n\ k = \frac{n!}{k!(n\text{-}k)!}$$

$$= \frac{n!}{k!(n\text{-}k)!} = \frac{(n\text{-}1)!n}{(k\text{-}1)!k((n\text{-}1)\text{-}(k\text{-}1))!} = \frac{n}{k}\text{comb}(n\text{-}1)(k\text{-}1)$$

- Λύση στη Haskell:

```
comb :: Int->Int->Int
comb n k
  | k == 0     = 1
  | True       = n * comb (n-1) (k-1) `div` k
```

- Αριθμοί Fibonacci

$$f_0 = f_1 = 1$$

$$n > 1 \Rightarrow f_n = f_{n\text{-}1} + f_{n\text{-}2}$$

- Λύση στη Haskell (μη πρωταρχική αναδρομή):

```
fib n
  | n <= 1    = 1
  | True      = fib(n-1)+fib(n-2)
```

- Πολύ κακή λύση: εκθετικός χρόνος. Θα επανέλθουμε

- Αριθμοί Fibonacci

$$f_0 = f_1 = 1$$

$$n > 1 \Rightarrow f_n = f_{n\text{-}1} + f_{n\text{-}2}$$

- Λύση στη Haskell (μη πρωταρχική αναδρομή):

```
fib n
  | n <= 1    = 1
  | True      = fib(n-1)+fib(n-2)
```

- Πολύ κακή λύση: εκθετικός χρόνος. Θα επανέλθουμε

- Αριθμοί Fibonacci

$$f_0 = f_1 = 1$$

$$n > 1 \Rightarrow f_n = f_{n\text{-}1} + f_{n\text{-}2}$$

- Λύση στη Haskell (μη πρωταρχική αναδρομή):

```
fib n
  | n <= 1    = 1
  | True      = fib(n-1)+fib(n-2)
```

- Πολύ κακή λύση: εκθετικός χρόνος. Θα επανέλθουμε

- Αμοιβαία αναδρομή: εκφράζει κάθε ροή ελέγχου
- Μετασχηματισμοί προγράμματος στον Σ.Π.
- Γράψτε μία συνάρτηση `power` που να υψώνει ένα θετικό ακέραιο σε έναν άλλο, χωρίς να χρησιμοποιεί την πράξη `^`
- Πρώτη προσέγγιση:

```
power :: Int->Int->Int
power x y
  | y == 0    = 1
  | True      = x * power x (y-1)
```

- Αμοιβαία αναδρομή: εκφράζει κάθε ροή ελέγχου
- Μετασχηματισμοί προγράμματος στον Σ.Π.
- Γράψτε μία συνάρτηση `power` που να υψώνει ένα θετικό ακέραιο σε έναν άλλο, χωρίς να χρησιμοποιεί την πράξη `^`
- Πρώτη προσέγγιση:

```
power :: Int->Int->Int
power x y
  | y == 0    = 1
  | True      = x * power x (y-1)
```

- Αμοιβαία αναδρομή: εκφράζει κάθε ροή ελέγχου
- Μετασχηματισμοί προγράμματος στον Σ.Π.
- Γράψτε μία συνάρτηση `power` που να υψώνει ένα θετικό ακέραιο σε έναν άλλο, χωρίς να χρησιμοποιεί την πράξη `^`
- Πρώτη προσέγγιση:

```
power :: Int->Int->Int
power x y
  | y == 0     = 1
  | True       = x * power x (y-1)
```

- Λογαριθμικός χρόνος: για άρτιο `y` μπορούμε να προχωρούμε πιο γρήγορα

```
power x y
 | y == 0     = 1
 | True       = x * power x (y-1)
| y `mod` 2 == 0      = power_even_non_zero_y x y
 | True                = power_odd_y x y
power_even_non_zero_y x y
   = power (x*x) (y `div` 2)
power_odd_y x y = x * power x (y-1)
```

- Λογαριθμικός χρόνος: για άρτιο `y` μπορούμε να προχωρούμε πιο γρήγορα

```
power x y
  | y == 0     = 1
  | True       = x * power x (y-1)
  | y `mod` 2 == 0      = power_even_non_zero_y x y
  | True                = power_odd_y x y
power_even_non_zero_y x y
    = power (x*x) (y `div` 2)
power_odd_y x y = x * power x (y-1)
```

- Άρτιο `y` και `y>0` $\Rightarrow$ `y‘div‘2>0`

```
power x y
  | y == 0                = 1
  | y ‘mod‘ 2 == 0        = power_even_non_zero_y x y
  | True                  = power_odd_y x y
power_even_non_zero_y x y
    = power  (x*x) (y ‘div‘ 2)
power_odd_y x y = x * power x (y-1)
```

- Άρτιο y και y>0 ⇒ y‘div‘2>0

```
power x y
  | y == 0                  = 1
  | y `mod` 2 == 0          = power_even_non_zero_y x y
  | True                    = power_odd_y x y
  | True                    = power_non_zero_y x y
power_non_zero_y x y
  | y `mod` 2 == 0  = power_even_non_zero_y x y
  | True            = power_odd_y x y
power_even_non_zero_y x y
    =  power power_non_zero_y  (x*x) (y `div` 2)
power_odd_y x y = x * power x (y-1)
```

- Περιττό `y` $\Rightarrow$ άρτιο `y-1`

```
power x y
 | y == 0                = 1
 | True                  = power_non_zero_y x y
power_non_zero_y x y
 | y `mod` 2 == 0   = power_even_non_zero_y x y
 | True             = power_odd_y x y
power_even_non_zero_y x y
   = power_non_zero_y (x*x) (y `div` 2)
power_odd_y x y = x * power  x (y-1)
```

- Περιττό `y` $\Rightarrow$ άρτιο `y-1`

```
power x y
 | y == 0                = 1
 | True                  = power_non_zero_y x y
power_non_zero_y x y
 | y `mod` 2 == 0   = power_even_non_zero_y x y
 | True             = power_odd_y x y
power_even_non_zero_y x y
   = power_non_zero_y (x*x) (y `div` 2)
power_even_y x y
 | y == 0   = 1
 | True     = power_even_non_zero_y x y
power_odd_y x y = x *  power power_even_y x (y-1)
```

- Καλύτερος έλεγχος στην αρχή: `y`mod`2 == 0`

```
power x y
 | y == 0                = 1
 | True                  = power_non_zero_y x y
power_non_zero_y x y
 | y `mod` 2 == 0  = power_even_non_zero_y x y
 | True             = power_odd_y x y
power_even_non_zero_y x y
   = power_non_zero_y (x*x) (y `div` 2)
power_even_y x y
 | y == 0  = 1
 | True    = power_even_non_zero_y x y
power_odd_y x y = x *  power_even_y x (y-1)
```

- Καλύτερος έλεγχος στην αρχή: `y`mod`2 == 0`

```
power x y
 | y == 0                = 1
 | True                  = power_non_zero_y x y
 | y `mod` 2 == 0  = power_even_y x y
 | True                  = power_odd_y x y
power_non_zero_y x y
 | y `mod` 2 == 0  = power_even_non_zero_y x y
 | True                  = power_odd_y x y
power_even_non_zero_y x y
   = power_non_zero_y (x*x) (y `div` 2)
power_even_y x y
 | y == 0  = 1
 | True    = power_even_non_zero_y x y
power_odd_y x y = x *  power_even_y x (y-1)
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
  | n==0      = 0
  | True      = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`
- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
  | n==0        = 0
  | True        = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`

- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
 | n==0       = 0
 | True       = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`

- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
  | n==0        = 0
  | True        = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`

- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
  | n==0        = 0
  | True        = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`
- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - I

- Παράδειγμα για: συναρτήσεις ανώτερης τάξης, currying, σύνθεση συναρτήσεων κτλ.
- Γράψτε πρόγραμμα που υπολογίζει $\sum_{i=1}^{n} f\ i$
- Πρόγραμμα Haskell:

```
sumf :: (Int->Int)->Int->Int
sumf f n
  | n==0        = 0
  | True        = f n + sumf f (n-1)
```

- Υπολογισμός $\sum_{i=1}^{n} i$ με currying:

```
sumupto :: Int->Int
sumupto n = sumf id
```

  - συνάρτηση ταυτότητας: `id x = x`
- Υπολογισμός $\sum_{i=1}^{10} i^2$:

```
sumf (^2) 10
```

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0   = 0
  | True             = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0   = 0
  | True             = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0   = 0
  | True             = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

```
sumf (fromIntegral.floor.log) 10
```

- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0 = 0
  | True           = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

```
sumf (fromIntegral.floor.log) 10
```

- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0 = 0
  | True           = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0 = 0
  | True           = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

Παραδείγματα
Υπολογισμός Αθροίσματος Συνάρτησης - II

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
  | n `mod` 2 == 0    = 0
  | True              = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

- Υπολογισμός $\sum_{i=1}^{10} \log i$:
  `sumf log 10`
  - λάθος τύπου: `log :: Float->Float`
- Υπολογισμός $\sum_{i=1}^{10} \lfloor \log i \rfloor$:
  `sumf (floor.log) 10`
  - λάθος τύπου: `floor.log :: Float->Int`

  `sumf (fromIntegral.floor.log) 10`
- Υπολογισμός $\sum_{i=1,\ i\ \text{περιττός}}^{10} i^2$

```
filterEven :: Int->Int
filterEven n
 | n `mod` 2 == 0    = 0
 | True              = n
```

Ερώτηση: `sumf ((^2).filterEven) 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:

```
inner i j = (i+j)^2
```

- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:

```
outer i = sumf (inner i) i
```

- Τελική ερώτηση: `sumf outer 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:

```
inner i j = (i+j)^2
```

- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:

```
outer i = sumf (inner i) i
```

- Τελική ερώτηση: `sumf outer 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:
  `inner i j = (i+j)^2`
- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:
  `outer i = sumf (inner i) i`
- Τελική ερώτηση: `sumf outer 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:
  `inner i j = (i+j)^2`
- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:
  `outer i = sumf (inner i) i`
- Τελική ερώτηση: `sumf outer 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:
  `inner i j = (i+j)^2`
- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:
  `outer i = sumf (inner i) i`
- Τελική ερώτηση: `sumf outer 10`

- Υπολογισμός: $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$
- Αρχίζουμε από μέσα: $(i+j)^2$:
  `inner i j = (i+j)^2`
- Επόμενος υπολογισμός: $\sum_{j=1}^{i} (i+j)^2$:
  `outer i = sumf (inner i) i`
- Τελική ερώτηση: `sumf outer 10`

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης

Επισκόπηση

Σε αυτή τη διάλεξη είδαμε

- Τα βασικά χρήσης ενός διερμηνέα Haskell
  - εκφράσεις, βασικούς τύπους, τελεστές, προτεραιότητα και προσεταιριστικότητα
  - προτάσεις, καθορισμούς τύπου και ορισμούς
- Συναρτήσεις:
  - τύπους
  - ορισμούς με τυπικές παραμέτρους
  - currying, σύνθεση, λύση συγκρούσεων ονομάτων
  - ενθεματική/προθεματική σύνταξη
  - φρουρούς
  - συναρτήσεις ανώτερης τάξης
  - αναδρομή
- Παραδείγματα:
  - χρήση (μη) πρωταρχικής αναδρομής
  - χρήση αμοιβαίας αναδρομής και μετασχηματισμούς
  - χρήση currying, σύνθεσης, συναρτήσεων ανώτερης τάξης