Συμβολικές Γλώσσες Προγραμματισμού

Ενότητα 7: Διαδικασιακός Προγραμματισμός

Νικόλαος Καραμπετάκης

Τμήμα Μαθηματικών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

1. Εντολές ανάθεσης.
2. Εντολές συνθήκης (If, Which, Switch).
3. Εντολές επανάληψης (Do, For, While).
4. Ομαδοποίηση εντολών - Διαδικασίες.
5. Άλλες εντολές ελέγχου ανακυκλώσεων (Break, Continue, Return, Abort, Goto, Label).
6. Παρακολούθηση της ροής αλλά και του χρόνου εκτέλεσης του προγράμματος.

Σκοποί Ενότητας

- Μελέτη των εντολών ανάθεσης, συνθήκης και επανάληψης.
- Μελέτη των διαδικασιών.

Εντολές ανάθεσης (1)

Απλή ανάθεση τιμής σε μεταβλητή (ες)

```
x = 1
```

```
1
```

```
x = y = 2
```

```
2
```

Αναζήτηση τιμών από τον χρήστη μέσω ενός διαλογικού παραθύρου

```
a = Input[]
```

```
5
```

```
a
```

```
5
```

Εντολές ανάθεσης (2)

ή έπειτα από ένα μήνυμα

```
Clear[x, y, a]
```

```
a = Input["Function F[x]:="]
```

```
2 + x
```

```
a
```

```
2 + x
```

Παρόμοια εκτός από εκφράσεις του *Mathematica* μπορούμε να διαβάσουμε και αλυσίδες χαρακτήρων χρησιμοποιόντας τις InputString[] και InputString["prompt"].

Εντολές συνθήκης

Οι εντολές συνθήκης που έχουμε στο *Mathematica* είναι οι

- If.
- Which.
- Switch.

Ας δούμε πρώτα την σύνταξη της εντολής If.

Εντολές συνθήκης (η εντολή If) (1)

```
? If
```

"If[condition, t, f] gives t if condition evaluates to True, and f if it evaluates to False. If[condition, t, f, u] gives u if condition evaluates to neither True nor False."

Συνεπώς στην περίπτωση που η condition είναι αληθής εκτελείται η πρόταση t διαφορετικά εκτελείται η πρόταση f και αν η συνθήκη δεν είναι ούτε αληθής ούτε ψευδής διότι πιθανόν τα ορίσματα της δεν έχουν αρχικά οριστεί τότε εκτελείται η πρόταση u.

Εντολές συνθήκης (η εντολή If) (2)

```
Clear[x, y, a]
```

```
f[x_] := If[x > 0, -x + 1, x^2 - 1]
```

```
f[2]
```

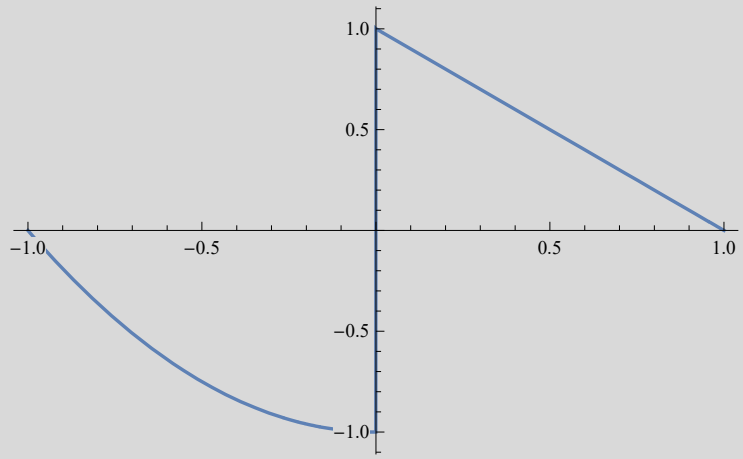
```
-1
```

```
f[-2]
```

```
3
```

Εντολές συνθήκης (η εντολή If) (3)

```
Plot[f[x], {x, −1, 1}]
```




```
If[x == y, x, y, z]
```

```
z
```

```
If[x === y, x, y]
```

```
y
```

Εντολές συνθήκης (η εντολή If) (4)

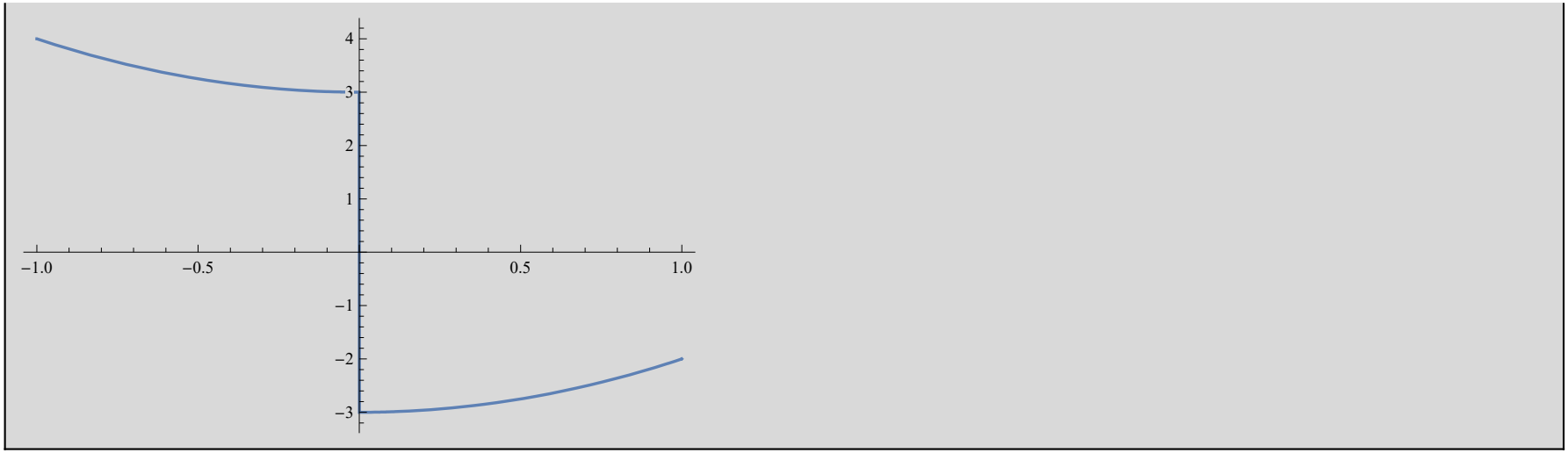
Άσκηση. Προσπάθησε να ορίσεις την συνάρτηση

$$f(x) = \begin{pmatrix} x^2 - 3 & x > 0 \\ x^2 + 3 & x \leq 0 \end{pmatrix}$$

και να υπολογίσεις τα σημεία f[-2],f[2].

```
f[x_] := If[x > 0, x^2 - 3, x^2 + 3]
```

```
Plot[f[x], {x, -1, 1}]
```



Εντολές συνθήκης (η εντολή If) (5)

```
f[-2]
f[2]
```

```
7
```

```
1
```

Εντολές συνθήκης (η εντολή Which)

```
? Which
```

```
Which[Subscript[test, 1], Subscript[value, 1], Subscript[test, 2], Subscript[value, 2], …]
 evaluates each of the Subscript[test, i] in turn,
returning the value of the Subscript[value, i] corresponding to the first one that yields True.  >>
```

Η εντολή Which ελέγχει λοιπόν τα παραπάνω test1,test2,... και στο πρώτο τεστ, έστω testi που θα βρει αληθές αποτέλεσμα θα εκτελέσει την valuei.

```
Which[2 > 3, 1, 1 > 4, 2, 2 == 2, 3]
```

```
3
```

```
Which[2 > 3, 1, 1 < 4, 2, 2 == 2, 3]
```

```
2
```

Άσκηση 1 (η εντολή Which) (1)

Άσκηση. Το καρτεσιανό επίπεδο συντεταγμένων χωρίζεται σε 4 τεταρτημόρια. Να γραφεί συνάρτηση που θα δέχεται ως είσοδο τις συντεταγμένες ενός σημείου (x,y) και θα απαντάει με 1,2,3,4 αν το σημείο βρίσκεται στο 1ο, 2ο, 3ο ή 4ο τεταρτημόριο. Αν το σημείο είναι το (0,0) να απαντάει με 0. Τέλος αν ανήκει στον άξονα των y (x=0) να απαντάει με -2 ενώ αν ανήκει στον άξονα των x (y=0) να απαντάει με -1.

```
f[x__, y__] := Which[
    x == 0 && y == 0,  0,
    y == 0,  −1,
    x == 0, −2,
    x > 0 && y > 0, 1,
    x < 0 && y > 0, 2,
    x < 0 && y < 0, 3,
    x > 0 && y < 0, 4]
```

Άσκηση 1 (η εντολή Which) (2)

```
f[0, 0]
f[1, 0]
f[0, 1]
f[1, 1]
f[−1, 1]
f[−1, −1]
f[1, −1]
```

```
0
```

```
−1
```

```
−2
```

```
1
```

```
2
```

```
3
```

```
4
```

Άσκηση 2 (η εντολή Which) (1)

Άσκηση. Να γραφεί συνάρτηση που θα δέχεται έναν ακέραιο αριθμό και θα επιστρέφει fail, good, very good, excellent αν ο αριθμός είναι αντίστοιχα από 0-4, 5-6, 7-8, 9-10.

```
Clear[f]
```

```
f[x__Integer] := Which[
    x ≥ 0 && x ≤ 4, "fail",
    x ≥ 5 && x ≤ 6, "good",
    x ≥ 7 && x ≤ 8, "very good",
    x ≥ 9 && x ≤ 10, "excellent",
    x < 0 || x > 10, "error"]
```

Άσκηση 2 (η εντολή Which) (2)

```
f[11]
```

```
error
```

```
f[9]
```

```
excellent
```

```
f[2.1]
```

```
f[2.1]
```

Άσκηση 2 (η εντολή Which) (3)

Άλλος τρόπος όπου με την /; περιορίζουμε το όρισμα να είναι ακέραιος στο διάστημα [0,10].

```
Clear[f]
```

```
f[x__ /; IntegerQ[x] && x ≥ 0 && x ≤ 10] :=
  Which[
    x ≥ 0 && x ≤ 4, "fail",
    x == 5 || x == 6, "good",
    x == 7 || x == 8, "very good",
    x == 9 || x == 10, "excellent"]
```

Άσκηση 2 (η εντολή Which) (4)

```
f[-2]
```

```
f[-2]
```

```
f[5]
```

```
good
```

```
f[1.1]
```

```
f[1.1]
```

Εντολές συνθήκης (η εντολή Switch) (1)

```
? Switch
```

```
Switch[expr, Subscript[form, 1], Subscript[value, 1], Subscript[form, 2], Subscript[value, 2], …] evaluates expr,
then compares it with each of the Subscript[form, i] in turn,
evaluating and returning the Subscript[value, i] corresponding to the first match found.  >>
```

Η εντολή Switch υπολογίζει αρχικά την expr και στη συνέχεια ελέγχει με ποια από τις παραστάσεις form1, form2,… είναι ίση. Για την πρώτη ίση έκφραση που θα βρεί, έστω formi, θα υπολογίσει την αντίστοιχη έκφραση valuei.

Η εντολή Head (1)

Όλες οι παραστάσεις στο *Mathematica* αντιστοιχούν σε συναρτήσεις. Παρακάτω βλέπουμε σε τι είδους συνάρτηση αντιστοιχεί το σύμβολο της πρόσθεσης, του πολλαπλασιασμού και της λίστας.

```
?Head
```

```
Head[expr] gives the head of expr.  >>
```

```
Head[a + b]
Head[a * b]
Head[{a, b}]
```

```
Plus
```

```
Times
```

```
List
```

Η παρακάτω συνάρτηση, πρώτα υπολογίζει το είδος της συνάρτησης που αντιστοιχεί στο όρισμα και στη συνέχεια αν είναι πρόσθεση επιστρέφει το ίδιο το όρισμα, αν είναι πολλαπλασιασμός επιστρέφει το τετράγωνο του ορίσματος και τέλος αν είναι λίστα επιστρέφει την λίστα με αντίστροφη σειρά.

Η εντολή Head (2)

```
f[x__] := Switch[Head[x],
    Plus, x,
    Times, x^2,
    List, Reverse[x]]
```

```
f[a + b]
f[a * b]
f[{a, b}]
```

$$a + b$$

$$a^2\ b^2$$

$$\{b,\ a\}$$

Άσκηση 3 (1)

Άσκηση 3. (Θεωρία Παιγνίων) Σε ένα παιχνίδι μεταξύ δύο ατόμων, η βαθμολογία έχει ως εξής :
α) Αν το άτομο Α παίξει την κίνηση 1 όταν το άτομο Β έπαιξε την κίνηση 1 τότε το άτομο Α κερδίζει 1 βαθμό,
β) Αν το άτομο Α παίξει την κίνηση 2 όταν το άτομο Β έπαιξε την κίνηση 1 τότε το άτομο Α χάνει 1 βαθμό,
γ) Αν το άτομο Α παίξει την κίνηση 1 όταν το άτομο Β έπαιξε την κίνηση 2 τότε το άτομο Α κερδίζει 2 βαθμούς,
δ) Αν το άτομο Α παίξει την κίνηση 2 όταν το άτομο Β έπαιξε την κίνηση 2 τότε το άτομο Α χάνει 2 βαθμούς,
Να γραφεί συνάρτηση που θα υπολογίζει το κέρδος του ατόμου Α ξέροντας τις κινήσεις των Α και Β.

Άσκηση 3 (2)

```
f[x__, y__] := Switch[{x, y},
     {1, 1}, 1,
     {2, 1}, −1,
     {1, 2}, 2,
     {2, 2}, −2]
```

Προσομοίωση τυχαίου παιξίματος για τον πάικτη 1 και 2

```
f[Random[Integer, {1, 2}], Random[Integer, {1, 2}]]
```

```
-2
```

```
Sum[f[Random[Integer, {1, 2}], Random[Integer, {1, 2}]],
 {i, 1, 1000000}]
```

```
2276
```

Εντολές επανάληψης

Οι εντολές ανακύκλωσης είναι οι Do, While και For.
Χρησιμοποιούνται όταν θέλω να επαναλάβω ένα σύνολο εντολών έναν συγκεκριμένο αριθμό φορών (Do, For) ή εως ότου μια συνθήκη ικανοποιηθεί (While).

Εντολές επανάληψης (η εντολή Do) (1)

```
? Do
```

```
Do[expr, n] evaluates expr n times.Do[expr, {i, Subscript[i, max]}] evaluates expr with the variable i successively taking on
  the values 1 through Subscript[i, max] (in steps of 1).Do[expr, {i, Subscript[i, min], Subscript[i, max]}] starts with i =
 Subscript[i, min].Do[expr, {i, Subscript[i, min], Subscript[i, max], di}] uses steps
  di.Do[expr, {i, {Subscript[i, 1], Subscript[i, 2], …}}] uses the successive values Subscript[i, 1],
Subscript[i, 2], ….Do[expr, {i, Subscript[i, min], Subscript[i, max]}, {j, Subscript[j, min], Subscript[j, max]}, …]
  evaluates expr looping over different values of j etc.for each i.  >>
```

Επαναλαμβάνει την expr η οποία μπορεί να είναι μια έκφραση ή ένα σύνολο εντολών, καθώς το i παίρνει τιμές από το 1 έως το imax ή από το imin έως το imax (με πιθανό βήμα di).

Εντολές επανάληψης (η εντολή Do) (2)

Ας δούμε τους 9 πρώτους αριθμούς

```
Do[Print[Prime[i]], {i, 1, 9}]
```

```
2
3
5
7
11
13
17
19
23
```

ή μπορούμε να πάρουμε τα στοιχεία και να τα προσθέσουμε

```
s = 0; Do[s = s + Prime[i], {i, 1, 9}]; s
```

```
100
```

Εντολές επανάληψης (η εντολή Do) (3)

ή μπορούμε να πάρουμε τα στοιχεία σε λίστα και στη συνέχεια να τα εκτυπώσουμε

```
s = {}; Do[s = Join[s, {Prime[i]}], {i, 1, 9}]; s
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23}
```

Εντολές επανάληψης (η εντολή Do) (4)

Ας υπολογίσουμε 5 τυχαίους ακέραιους αριθμούς από το 1 ως το 49 και έναντυχαίο αριθμό από το 1 έως το 20

```
Do[Print[Random[Integer, {1, 49}]], {5}]
Random[Integer, {1, 20}]
```

```
3
30
32
27
24
```

```
20
```

Εντολές επανάληψης (η εντολή Do) (5)

Όμοια μπορούμε να πάρουμε τα παραπάνω νούμερα σε λίστα (πως ;)

```
s = {};
Do[s = Join[s, {Random[Integer, {1, 49}]}], {5}];
s = Join[s, {Random[Integer, {1, 20}]}]; s
```

```
{23, 16, 11, 19, 40, 17}
```

Εντολές επανάληψης (η εντολή Do) (6)

Μπορούμε να χρησιμοποιήσουμε και διπλές ανακυκλώσεις με την χρήση δύο do. Στο επόμενο παράδειγμα προσπαθούμε να υπολογίσουμε τους πυθαγόρειους αριθμούς (x,y,z) με 1‹=x,y,z‹=10.

```
Do[
  Do[
    Do[
      If[x^2 + y^2 == z^2, Print["(", x, ",", y, ",", z, ")"]]
      , {z, 1, 10}]
    , {y, 1, 10}]
  , {x, 1, 10}]
```

```
(3,4,5)
(4,3,5)
(6,8,10)
(8,6,10)
```

Εντολές επανάληψης (η εντολή Do) (7)

```
Do[Print[x, ",", y], {x, 1, 2}, {y, 1, 3}]
```

```
1,1
1,2
1,3
2,1
2,2
2,3
```

```
Do[If[x^2 + y^2 == z^2, Print["(", x, ",", y, ",", z, ")"]],
 {x, 1, 10}, {y, 1, 10}, {z, 1, 10}]
```

```
(3,4,5)
(4,3,5)
(6,8,10)
(8,6,10)
```

Εντολές επανάληψης (η εντολή Do) (8)

ή ακόμα καλύτερα για να αποφύγουμε επαναλήψεις τριάδων

```
Do[
  Do[
    Do[
      If[x^2 + y^2 == z^2, Print["(", x, ",", y, ",", z, ")"]]
      , {z, y + 1, 10}]
    , {y, x + 1, 9}]
  , {x, 1, 8}]
```

```
(3,4,5)
(6,8,10)
```

```
Do[If[x^2 + y^2 == z^2, Print["(", x, ",", y, ",", z, ")"]],
 {x, 1, 8}, {y, x + 1, 9}, {z, y + 1, 10}]
```

```
(3,4,5)
(6,8,10)
```

Ασκήσεις (η εντολή Do) (1)

Άσκηση 1. Να υπολογιστεί ο 10ος όρος της ακολουθίας

$$a_n = \frac{1}{2}\left(a_{n-1} + \frac{2}{a_{n-1}}\right),\ a_0 = 1$$

Άσκηση 2. Να υπολογιστούν οι 10 πρώτοι όροι της ακολουθίας Fibonacci

$$a_n = a_{n-1} + a_{n-2},\ a_1 = 1,\ a_2 = 1$$

Ασκήσεις (η εντολή Do) (2)

Λύση άσκησης 1.

```
x0 = 1; Do[(x1 = 1/2 (x0 + 2/x0); x0 = x1), {9}]; x1 // N
```

```
1.41421
```

```
a = 1; Do[a = 1/2 (a + 2/a), {9}]; a // N
```

```
1.41421
```

Ασκήσεις (η εντολή Do) (3)

Λύση άσκησης 2.

```
s1 = 1; s1
s2 = 1; s2
Do[{s = s1 + s2, Print[s], s2 = s1, s1 = s},  {8}]
```

```
1
```

```
1
2
3
5
8
13
21
34
55
```

Ασκήσεις (η εντολή Do) (4)

ή αν θέλαμε μόνο τον νοστό όρο

```
Clear[f]
```

```
f[n__Integer] :=
  (s1 = 1;
    s2 = 1;
    Do[s = s1 + s2; s2 = s1; s1 = s,  {i, 1, n − 2}];
    s)
```

```
f[10]
```

```
55
```

```
Fibonacci[10]
```

```
55
```

Εντολές επανάληψης (η εντολή For) (1)

```
? For
```

```
For[start, test, incr, body] executes start, then repeatedly evaluates body and incr until test fails to give True.  >>
```

Ο μετρητής της ανακύκλωσης πέρνει αρχική τιμή με την start. Στη συνέχεια υπολογίζεται το σύνολο των εντολών body και αυξάνει το βήμα σύμφωνα με την incr έως ότου η συνθήκη test γίνει αληθής.
Ας υποθέσουμε ότι θέλουμε να εκτυπώσουμε τους 3 πρώτους αριθμούς.

```
For[i = 1, i ≤ 3, ++i, Print[Prime[i]]]
```

```
2
3
5
```

ή με την εντολή Do

```
Do[Print[Prime[i]], {i, 1, 3}]
```

```
2
3
5
```

τα αναπτύγματα της $(x+1)^n$

Ας δούμε τα αναπτύγματα της $(x+1)^n$ καθώς το n παίρνει τιμές 1,2,..,10

```
For[n = 1, n ≤ 10, ++n, Print[Expand[(x + 1)^n]]]
```

$1 + x$

$1 + 2\,x + x^2$

$1 + 3\,x + 3\,x^2 + x^3$

$1 + 4\,x + 6\,x^2 + 4\,x^3 + x^4$

$1 + 5\,x + 10\,x^2 + 10\,x^3 + 5\,x^4 + x^5$

$1 + 6\,x + 15\,x^2 + 20\,x^3 + 15\,x^4 + 6\,x^5 + x^6$

$1 + 7\,x + 21\,x^2 + 35\,x^3 + 35\,x^4 + 21\,x^5 + 7\,x^6 + x^7$

$1 + 8\,x + 28\,x^2 + 56\,x^3 + 70\,x^4 + 56\,x^5 + 28\,x^6 + 8\,x^7 + x^8$

$1 + 9\,x + 36\,x^2 + 84\,x^3 + 126\,x^4 + 126\,x^5 + 84\,x^6 + 36\,x^7 + 9\,x^8 + x^9$

$1 + 10\,x + 45\,x^2 + 120\,x^3 + 210\,x^4 + 252\,x^5 + 210\,x^6 + 120\,x^7 + 45\,x^8 + 10\,x^9 + x^{10}$

Εντολές επανάληψης (η εντολή For) (2)

Μπορούμε να έχουμε και την μια For μέσα στην άλλη όπως παρακάτω

```
Clear[x, y, z]
```

```
For[x = 1, x ≤ 8, ++x,
  For[y = x + 1, y ≤ 9, ++y,
    For[z = y + 1, z ≤ 10, ++z,
      If[x^2 + y^2 == z^2, Print["(", x, ",", y, ",", z, ")"]]
    ]
  ]
]
```

```
(3,4,5)
(6,8,10)
```

Εντολές επανάληψης (η εντολή For) (3)

Ας προσπαθήσουμε τώρα να υπολογίσουμε την ρίζα μιας συνάρτησης f[x] με την μέθοδο Newton

$$a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)},\ a_0 = ?$$

```
f[x_] := x^3 - 2
```

```
a0 = 1
```

```
1
```

```
a1 = N[a0 - f[a0] / f'[a0]]
```

```
-17.3333
```

```
a2 = N[a1 - f[a1] / f'[a1]]
```

```
-11.5533
```

Εντολές επανάληψης (η εντολή For) (4)

ή θα μπορούσαμε να ορίσουμε μια συνάρτηση nn που θα υπολογίζει τον επόμενο όρο της ακολουθίας

```
nn[a__, f__] := N[a - f[a] / f'[a]]
```

και να επαναλάβουμε ένα σύνολο φορών την εντολή αυτή με αρχική τιμή του a το a_0

```
Clear[a0]
```

```
a0 = 1; For[i = 1, i ≤ 10, ++i, a0 = nn[a0, f]]; a0
```

```
1.25992
```

```
N[2^(1/3)]
```

```
1.25992
```

Εντολές επανάληψης (η εντολή For) (5)

Θα μπορούσαμε και να τυπώσουμε όλους τους όρους της ακολουθίας για να δούμε τον ρυθμό σύγκλισης

```
a0 = 1;
```

```
For[i = 1, i ≤ 10, ++i, (a0 = nn[a0, f]; Print[a0])]
```

```
-17.3333
-11.5533
-7.69723
-5.12023
-3.38806
-2.20063
-1.32942
-0.509074
2.23306
1.6224
```

Ασκήση 4 (η εντολή For)

Άσκηση 4. Προσπαθήστε να υπολογίσετε το άθροισμα 1+3+5+7+...+1001.

```
s = 0; For[i = 1, i ≤ 1001, i = i + 2, s = s + i]; s
```

```
251001
```

```
s = 0; Do[s = s + i, {i, 1, 1001, 2}]; s
```

```
251001
```

```
Sum[i, {i, 1, 1001, 2}]
```

```
251001
```

Εντολές επανάληψης (η εντολή While) (1)

```
? While
```

```
While[test, body] evaluates test, then body, repetitively, until test first fails to give True.  >>
```

Πρώτα γίνεται ο έλεγχος της συνθήκης test. Όσο η συνθήκη είναι αληθής η πρόταση body εκτελείται. Θα πρέπει να δωθεί μεγάλη προσοχή ώστε τα ορίσματα που εμπλέκονται στη συνθήκη να έχουν πάρει τιμή πριν την While αλλά και να αλλάξουν τιμή μέσα στην Do. Συνήθως η While χρησιμοποιείται όταν δεν ξέρουμε εκ των προτέρων τον αριθμό των επαναλήψεων που θα οδηγήσουν στο αποτέλεσμα που θέλουμε.

Παρακάτω υπολογίζουμε το άθροισμα των αριθμών 1+3+5+...+1001.

```
s = 0; i = 1; While[i ≤ 1001, (s = s + i; i = i + 2)]; s
```

```
251001
```

Η παραπάνω υλοποίηση είδαμε ότι γινόταν εξίσου και με την εντολή Do και For.

Εντολές επανάληψης (η εντολή While) (2)

Ας υποθέσουμε ότι θέλουμε να υπολογίσουμε τον πρώτο αριθμό Fibonacci που διαιρείται με 100.

```
f1 = 1;
f2 = 1;
i = 3;
f = f1 + f2;
While[Mod[f, 100] ≠ 0,
      f1 = f2;
      f2 = f;
      f = f1 + f2;
      ++i];
f
i
```

```
9 969 216 677 189 303 386 214 405 760 200
```

```
150
```

Εντολές επανάληψης (η εντολή While) (3)

Πράγματι

```
Fibonacci[150]
```

```
9 969 216 677 189 303 386 214 405 760 200
```

Το πρόγραμμα θα μπορούσε να γραφτεί με χρήση της έτοιμης συνάρτησης Fibonacci

```
i = 1;
While[Mod[Fibonacci[i], 100] ≠ 0, ++i];
Fibonacci[i]
i
```

```
9 969 216 677 189 303 386 214 405 760 200
```

```
150
```

Fibonacci-Fortran (1)

Ας δούμε το ίδιο πρόγραμμα σε Fortran τι αποτέλεσμα θα βγάλει

```
program fibonacci
implicit none
! Variables
INTEGER*4    :: f,f1,f2,i
! Body of fibonacci
f1 = 1;  f2 = 1;  i = 3 ; f = f1 + f2;
Do While (Mod(f,100)/=0)
        f1 = f2 ; f2 = f ; f = f1 + f2 ; i=i+1
End Do
Print*,f,i
end program fibonacci
```

```
  708252800          96
Press any key to continue
```

Τι παρατηρείται ; Η διαφορά στις τιμές οφείλεται στο περιορισμένο εύρος των τιμών των ακεραίων Integer*4 (μέγιστος ακέραιος 2^31-1). Μια διόρθωση στον τύπο των ακεραίων Integer*8 (μέγιστος ακέραιος 2^64-1) δεν θα έδινε πάλι το σωστό αποτέλεσμα

```
  8284360270132553400                 522
Press any key to continue
```

Fibonacci-Fortran (2)

Ας δούμε το ίδιο πρόγραμμα σε Fortran τι αποτέλεσμα θα βγάλει

```
program fibonacci
implicit none
! Variables
INTEGER*4    :: f,f1,f2,i
! Body of fibonacci
f1 = 1;  f2 = 1;  i = 3 ; f = f1 + f2;
Do While (Mod(f,100)/=0)
        f2 = f1 ; f1 = f ; f = f1 + f2 ; i=i+1
End Do
Print*,f,i
end program fibonacci
```

```
  708252800          96
Press any key to continue
```

Τι παρατηρείται ; Η διαφορά στις τιμές οφείλεται στο περιορισμένο εύρος των τιμών των ακεραίων Integer*4 (μέγιστος ακέραιος 2^31-1). Μια διόρθωση στον τύπο των ακεραίων Integer*8 (μέγιστος ακέραιος 2^64-1) δεν θα έδινε πάλι το σωστό αποτέλεσμα

```
  8284360270132553400                522
Press any key to continue
```

Ασκήση 5 (1)

Άσκηση 5. Να δείξετε ότι δεν ισχύει η εικασία ότι ο αριθμός $2^p - 1$ όπου p πρώτος αριθμός, είναι πάντα πρώτος αριθμός.

```
PrimeQ[2^Prime[1] - 1]
```

```
True
```

```
PrimeQ[2^Prime[2] - 1]
```

```
True
```

και θα συνεχίσουμε με παρόμοιες εντολές έως ότου να πάρουμε απάντηση False.

Συνεπώς θα μπορούσαμε να γράψουμε το πρόγραμμα ως εξής

Ασκήση 5 (2)

```
i = 1;
While[PrimeQ[2^Prime[i] - 1],
  ++i]
Print[i, "th prime=", Prime[i]]
```

```
5th prime=11
```

```
FactorInteger[2^Prime[5] - 1]
```

```
{{23, 1}, {89, 1}}
```

Ομαδοποίηση εντολών - Διαδικασίες

Ας δούμε ξανά την συνάρτηση που γράψαμε για τον υπολογισμό της ακολουθίας Fibonacci

```
Clear[f]
```

```
f[n__Integer] :=
  (s1 = 1;
    s2 = 1;
    Do[{s = s1 + s2, s2 = s1, s1 = s},  {i, 1, n – 2}];
    s)
```

```
f[5]
```

```
5
```

Η εντολή Module (1)

Αν όμως τώρα χρησιμοποιήσουμε σε κάποια έκφραση τα s1,s2,s,i αυτά θα έχουν ήδη πάρει τιμή μέσα από τη συνάρτηση

```
x = s + 1
```

```
6
```

Παρατηρούμε λοιπόν ότι οι μεταβλητές που χρησιμοποιούμε στις συναρτήσεις που γράφουμε επηρεάζουν άμεσα και το υπόλοιπο μέρος του προγράμματος\session μας. Οι μεταβλητές αυτές ονομάζονται καθολικές ή global. Προκειμένου οι μεταβλητές αυτές να μην επηρεάσουν τις υπόλοιπες εντολές (εκτός συνάρτησης) θα πρέπει να τις ορίσουμε ως τοπικές (local). Αυτό το επιτυγχάνουμε με την χρήση της Module και Block.

```
? Module
```

```
Module[{x, y, …}, expr] specifies that occurrences of the symbols x, y,
… in expr should be treated as local.Module[{x = Subscript[x, 0], …}, expr] defines initial values for x, ….  >>
```

Η εντολή Module (2)

```
Clear[f, s, s1, s2, i]
```

```
fib[n__Integer] := Module[{s1, s2, s, i},
    If[n == 1, s = 1,
      If [n == 2, s = 1,
        s1 = 1;
        s2 = 1;
        Do[{s = s1 + s2, s2 = s1, s1 = s},  {i, 1, n – 2}]]];
    s]
```

```
fib[1]
```

```
1
```

```
fib[2]
```

```
1
```

Η εντολή Module (3)

```
fib[10]
```

```
55
```

```
fib[n_Integer] := Module[{s1 = 1, s2 = 1, s, i},
  If[n == 1, s = 1,
   If [n == 2, s = 1,
    Do[{s = s1 + s2, s2 = s1, s1 = s}, {i, 1, n - 2}]]];
  s]
```

```
fib[10]
```

```
55
```

```
x = s + 1
```

```
1 + s
```

Η εντολή Block (1)

Εκτός της Module για την σύνταξη διαδικασιών, έχουμε και την Block.

```
? Block
```

```
Block[{x, y, …}, expr] specifies that expr is to be evaluated with local values for the symbols x,
y, ….Block[{x = Subscript[x, 0], …}, expr] defines initial local values for x, ….  >>
```

Όπως παρατηρούμε παραπάνω η σύνταξη της Block είναι η ίδια με την Module. Παρόλα αυτά η διαφορά έγκειται στην λεξική εμβέλεια-ορατότητα των μεταβλητών που έχουμε στην Module και της δυναμικής εμβέλειας-ορατότητας μεταβλητών που έχουμε στην Block.
Για να δούμε την διαφορά της Block με την Module θεωρείστε το παρακάτω παράδειγμα :

Η εντολή Block (2)

```
Clear[i, a, m]
```

```
m = i^2
```

```
i^2
```

```
Block[{i = a}, i + m]
```

```
a + a^2
```

```
Module[{i = a}, i + m]
```

```
a + i^2
```

Παρατηρούμε λοιπόν ότι στην Block το m δέχτηκε την τιμή i^2 από την προηγούμενη ανάθεση και παρόλο που η μεταβλητή i είναι τοπική στο Block και δεν θα έπρεπε να έχει σχέση με την εξωτερική μεταβλητή i (όπως στην Module), έγινε η αντικατάσταση του καθολικού i με την τιμή του τοπικού i που είναι a. Μέσα στην Block λοιπόν η καθολική μεταβλητή i αποκτά το ίδιο περιεχόμενο με αυτό της τοπικής μεταβλητής i.

Η εντολή With (1)

Όπως είδαμε και από προηγούμενα παραδείγματα μια σειρά από εκφράσεις για υπολογισμό μπορούν να γραφούν με ένα ερωτηματικό μεταξύ τους όπως expr1;expr2;expr3;.... Τέλος αν θέλουμε να υπολογίσουμε μια έκφραση για συγκεκριμένες τιμές μεταβλητών, τότε μπορούμε να χρησιμοποιήσουμε την With.

```
? With
```

```
With[{x = Subscript[x, 0], y = Subscript[y, 0], …}, expr] specifies that in expr occurrences of the symbols x,
y, … should be replaced by Subscript[x, 0], Subscript[y, 0], ….  >>
```

$$\text{With}\left[\{t = 1\},\ t^2 + 1\right]$$

```
2
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Break) (1)

Η εντολή Break

```
? Break
```

Χρησιμοποιείται για την έξοδο από μια ανακύκλωση. Στο παρακάτω μέρος προγράμματος υπολογίζει τους Fibonacci αριθμούς από τον πρώτο έως και τον πρώτο που θα βρεί που είναι πολλαπλάσιο του 10. Μόλις βρει τον αριθμό αυτό θα σταματήσει το πρόγραμμα.

```
For[i = 1, i ≤ 100, ++i, If[(Mod[Fibonacci[i], 10] == 0 ), Break[]]];
i
Fibonacci[i]
```

```
15
```

```
610
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Break) (2)

```
For[i = 1, Mod[Fibonacci[i], 10] != 0, ++i];
i
Fibonacci[i]
```

```
15
```

```
610
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Continue)

Η εντολή Continue

```
? Continue
```

Παραπέμπει στην πλησιέστερη Do,For ή While, παραλείποντας το υπόλοιπο μέρος της τρέχουσας ανακύκλωσης. Στο παρακάτω μέρος προγράμματος αν βρούμε έναν άρτιο αριθμό πηγαίνουμε στην επόμενη τιμή της μεταβλητής i.

```
For[i = 1, i ≤ 5, ++i, If[Mod[i, 2] ≠ 0, Print[i], Continue[]]]
```

```
1
3
5
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Return)

Η εντολή Return

```
?Return
```

Έξοδος από την συνάρτηση και επιστροφή της συγκεκριμένης τιμής ή καμιάς τιμής αν Return[ ].

```
f[x__] := If[x > 0, Return[x + 1], Return[ ]]
```

```
f[−2]
```

```
f[3]
```

```
4
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Abort) (1)

Η εντολή Abort

```
? Abort
```

Χρησιμοποιείται αν θέλουμε να σταματήσουμε τον υπολογισμό μιας έκφρασης.
Στην μέθοδο Newton θα θέλαμε να σταματήσουμε την διαδικασία σύγκλισης αν έχει προηγουμένως γίνει η ανακύκλωση ένα συγκεκριμένο αριθμό φορών χωρίς να υπάρξει σύγκλιση.

Άλλες εντολές ελέγχου ανακυκλώσεων (Abort) (2)

```
Clear[f, a, a1, i]
```

```
f[x__] := x^2 + 3
```

```
a = 1;
i = 1;
a1 = N[a – f[a]/f'[a]];
While[Abs[a1 – a] > 0.0001,
      a = a1;
      a1 = N[a – f[a]/f'[a]];
      i = i + 1;
      If[i > 256, Abort[ ]];
   ];
a1
```

$2.00294 \times 10^{-12} + 1.73205\, i$

```
(* Προσπάθησε με a=1+I *)
```

Άλλες εντολές ελέγχου ανακυκλώσεων (Abort-Go to-Label) (3)

? Abort

Οι εντολές Goto και Label

? Goto

? Label

Ροή και χρόνος εκτέλεσης προγράμματος (1)

Παρακαλούθηση της ροής αλλά και του χρόνου εκτέλεσης του προγράμματος

```
? Trace
```

Η Trace μας βοηθάει να ελέγξουμε την ροή εκτέλεσης του προγράμματος μας.

```
Trace[(3 + 4)^2 (4 - 3)]
```

```
{{{3 + 4, 7}, 7^2, 49}, {4 - 3, 1}, 49 × 1, 49}
```

```
fac[n_Integer] :=
  If[n < 0, Return[],
    If[n == 0, 1, p = 1; Do[p = p * i, {i, 1, n}]; p]]
```

Ροή και χρόνος εκτέλεσης προγράμματος (2)

Trace[fac[3]]

```
{fac[3], If[3 < 0, Return[],
  If[3 == 0, 1, p = 1; Do[p = p i, {i, 1, 3}]; p]], {3 < 0, False},
 If[False, Return[], If[3 == 0, 1, p = 1; Do[p = p i, {i, 1, 3}]; p]],
 If[3 == 0, 1, p = 1; Do[p = p i, {i, 1, 3}]; p], {3 == 0, False},
 If[False, 1, p = 1; Do[p = p i, {i, 1, 3}]; p],
 p = 1; Do[p = p i, {i, 1, 3}]; p, {p = 1, 1},
 {Do[p = p i, {i, 1, 3}], {{{p, 1}, {i, 1}, 1 × 1, 1}, p = 1, 1},
  {{{p, 1}, {i, 2}, 1 × 2, 2}, p = 2, 2},
  {{{p, 2}, {i, 3}, 2 × 3, 6}, p = 6, 6}, Null}, {p, 6}, 6}
```

Trace[fac[4], p = __]

```
{{p = 1}, {{p = 1}, {p = 2}, {p = 6}, {p = 24}}}
```

Trace[fac[4], __ = __]

```
{{p = 1}, {{p = 1}, {p = 2}, {p = 6}, {p = 24}}}
```

Ροή και χρόνος εκτέλεσης προγράμματος (3)

Η εντολή Timing επιστρέφει τον χρόνο εκτέλεσης μιας συνάρτησης.

```
? Timing
```

```
Timing[FindMinimum[x^2 - 3, {x, 1}]]
```

```
{0., {-3., {x → 0.}}}
```

```
? TimeConstrained
```

Γενικές Ασκήσεις (1)

Άσκηση 1. Να γραφεί συνάρτηση που θα επιστρέφει 5 ανόμοιους ακέραιους αριθμούς από το 1 ως το 49.

```
s = {};
q = Random[Integer, {1, 49}];
i = 1;
While[! MemberQ[q, s] && i ≤ 5,
  s = AppendTo[s, q]; q = Random[Integer, {1, 49}]; i = i + 1];
s
```

```
{33, 46, 20, 38, 37}
```

```
(* άλλος τρόπος *)
s = {};
While[Length[s] ≠ 5, s = Union[s, {Random[Integer, {1, 49}]}]];
s
```

```
{5, 23, 26, 33, 45}
```

Γενικές Ασκήσεις (2)

Άσκηση 2. Να γραφεί συνάρτηση που θα υπολογίζει την τιμή της

$$f(x) = \begin{cases} x - 1 & x \le 1 \\ x + 1 & 1 < x \le 2 \\ 2x + 1 & 2 < x \end{cases}$$

Να λυθεί το παραπάνω πρόβλημα κάνοντας χρήση α) της εντολής If, β) της εντολής Which.

Γενικές Ασκήσεις (3)

Άσκηση 3. (Θεωρία Παιγνίων) Σε ένα παιχνίδι μεταξύ δύο ατόμων, η βαθμολογία έχει ως εξής :
α) Αν το άτομο Α παίξει την κίνηση 1 όταν το άτομο Β έπαιξε την κίνηση 1 τότε το άτομο Α κερδίζει 1 βαθμό,
β) Αν το άτομο Α παίξει την κίνηση 2 όταν το άτομο Β έπαιξε την κίνηση 1 τότε το άτομο Α χάνει 1 βαθμό,
γ) Αν το άτομο Α παίξει την κίνηση 1 όταν το άτομο Β έπαιξε την κίνηση 2 τότε το άτομο Α κερδίζει 2 βαθμούς,
δ) Αν το άτομο Α παίξει την κίνηση 2 όταν το άτομο Β έπαιξε την κίνηση 2 τότε το άτομο Α χάνει 2 βαθμούς,
Αν θεωρήσουμε ότι οι κινήσεις των παικτών Α και Β είναι τυχαίες (δηλαδή παίρνουν τυχαία τις τιμές 1 ή 2), να υπολογιστεί το κέρδος του παίκτη Α μετά από 1000, 10000, 100000, 1000000 κινήσεις που έπαιξε. Να κάνετε χρήση της συνάρτησης υπολογισμού κέρδους του παίκτη Α, που περιγράψαμε παραπάνω.

Γενικές Ασκήσεις (4)

Άσκηση 4. Να γραφεί συνάρτηση που θα προσομοιώνει την παρακάτω διαδικασία : α) θα δημιουργεί μια λίστα a1 με τους φυσικούς αριθμούς 1,2,3,...,n (το n θα δίνεται ως είσοδος στη διαδικασία), β) στη συνέχεια θα δημιουργεί μια λίστα a2 που θα περιέχει τα στοιχεία της a1 εκτός από το 1ο, 3ο, 5ο,...,50ο, γ) ακολούθως θα δημιουργεί μια λίστα a3 που θα περιέχει τα στοιχεία της a2 εκτός από το 1ο, 3ο, 5ο,.... δ) η διαδικασία αυτή θα συνεχίζεται εως ότου μείνει μια λίστα με ένα μόνο στοιχείο, ε) η συνάρτηση θα επιστρέφει το στοιχείο αυτό.

```
f[n_Integer] := Module[{a1 = Table[i, {i, 1, n}], i},
  While[Length[a1] ≠ 1,
   a1 = Drop[a1, {1, Length[a1], 2}]];
  a1[[1]]]
```

```
f[100]
```

```
64
```

Γενικές Ασκήσεις (5)

```
g[n_] := Module[{i = 1},
  While[2^i ≤ n, i = i + 1];
  2^(i-1)
 ]
```

```
g[100]
```

```
64
```

Σημείωμα Αναφοράς

Διαθέσιμο από τη δικτυακή διεύθυνση:

```
http://eclass.auth.gr/courses/OCRS430/
```

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Τέλος ενότητας

Επεξεργασία: Αναστασία Γ. Γρηγοριάδου
Θεσσαλονίκη, Εαρινό εξάμηνο 2014-2015

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης