ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΤΗΣ ΓΛΩΣΣΑΣ JULIA

Αναστασία-Ευτέρπη Ψήτου

Στα πλαίσια του μαθήματος

Λογισμός Πινάκων και Εφαρμογές

Διδάσκουσα: Μαριλένα Μητρούλη

Εαρινό Εξάμηνο 2021

Julia

Εισαγωγή

Η Julia είναι μια high-level, high-performance, δυναμική γλώσσα προγραμματισμού.

Πρωτοεμφανίστηκε ως ιδέα το 2009 και ξεκινήσε να αναπτύσσεται το 2012 από τους ερευνητές του MIT's CSAIL (Computer Science and Artificial Intelligence Lab) ως demo (Julia 0.X), ενώ τον Αύγουστο του 2018 απέκτησε stable release (Julia 1.X), με την πιο πρόσφατη έκδοση του να είναι η Julia 1.5.3.(9/11/2020)

Η Julia ως general-purpose γλώσσα προγραμματισμού, που μπορεί να χρησιμοποιηθεί και για τη δημιουργία εφαρμογών, χρίζεται καταλλήλη για την υπολογιστική επιστημή και την αριθμητική ανάλυση.

Επιπρόσθετα χαρακτηριστικά:

- Technical
- Optionally typed
- Reproducible
- Composable
- Open source

Αξίζει να σημειωθεί ότι, η Julia 1.X αποδίδει συγκρίσιμες ταχύτητες με τη C++, με την υψηλή αποδοτικότητα και ευκολία χρήσης της Python και R, καθώς επίσης αναγνωρίζει βιβλιοθήκες από άλλες γλώσσες προγραμματισμού, όπως της Python, R, C, C++, Fortran και Java. Επιπλέον, χρησιμοποιεί εντατική αξιολόγηση, είναι garbage-collected και περιέχει αποτελεσματικές βιβλιοθηκές για υπολογισμούς floating-point, γραμμικής άλγεβρας, γεννήτριας τυχαίων αριθμών και αντιστοιχείας κανονικών εκφράσεων(regular expression matching).

Οδηγός Εγκατάστασης

Σύνδεσμοι Εγκατάστασης:

1. Julia standalone
2. Julia pro

Μερικά Editors και IDEs:

- Juno
- VS Code
- Jupyter
- NotePad++

Βασικές γνώσεις Julia

Script

Τα προγράμματα στην Julia αποθηκεύονται στη μορφή: **όνομα_προγράμματος.jl**

Αριθμοί και Αριθμητικοί τελεστές

- Πρόσθεση = `+`

In [1]:
```
2 + 5
```

Out[1]: 7

- Αφαίρεση = `-`

In [2]:
```
2 - 5
```

Out[2]: -3

- Πολλαπλασιασμός = `*`

In [3]:
```
2 * 5
```

Out[3]: 10

- Διαίρεση = `/`

In [4]:
```
2 / 5
```

Out[4]: 0.4

- Ύψωση σε κάποια δύναμη = `^`

In [5]:
```
2 ^ 5
```

Out[5]: 32

- Floor division = `\div TAB` (διαίρεση δύο αριθμών και στρογγυλοποίηση του αποτελέσματος προς στον μικρότερο ακέραιο.)

In [6]:
```
54 ÷ 5
```

Out[6]: 10

- Modulus = `%`

In [7]:
```
54 % 5
```

Out[7]: 4

Ιδιαίτεροι Αριθμοί

- Για να ξεχωρίσουμε ψηφία πολύ μεγάλων αριθμών χρησιμοποιούμε την " _ " (κάτω παύλα), δηλαδή για τον αριθμό 123.456, τον συμβολίζουμε ως `123_456` και όχι 123.456 ή 123,456.
- Ο αριθμός π ≃ 3.14 στην Julia καλείται ως: `\pi TAB` *(γράφουμε \pi και πατάμε TAB button)*
- Ο αριθμος ε > 0 στην Julia καλείται ως: `\varepsilon TAB` *(γράφουμε \varepsilon και πατάμε TAB button)*

Συμβολοσειρές

- Είναι της μορφής: `"Hello world!!"` ή `"""Hello world!!"""`

In [8]:
```
"Hello world!!"
```

Out[8]: "Hello world!!"

In [9]:
```
"""Hello wordl!!"""
```

Out[9]: "Hello wordl!!"

- Αποτελούνται απο χαρακτήρες του Unicode.

In [10]:
```
x = "Καλησπέρα και καλή βραδιά"
x[1]
```

Out[10]: 'K': Unicode U+039A (category Lu: Letter, uppercase)

- Μπορούν να αναγνωρίσουν τους αριθμητικούς τελεστές *και* ^, *όπου* στις συμβολοσειρές ερμηνεύεται ως ένωση συμβολοσειρών και ^ ως επανάληψη συμβολοσειράς.

In [11]:
```
"Hello" * "World"
```

Out[11]: "HelloWorld"

In [12]:
```
"Spam"^5
```

Out[12]: "SpamSpamSpamSpamSpam"

- Μπορούμε να επιλέξουμε συγκεκρίμενα τμήματα συμβολοσειρών.
  - x[n] = ο n-οστος χαρακτήρας της συμβολοσειράς του x.
  - x[n:m] = από τον n εως και τον m χαρακτήρα του x.

In [13]:
```
#x[1:5] = από τον 1ο εως και τον 5ο χαρακτήρα του x.
x = "Hello world!!"
x[1:5]
```

Out[13]: "Hello"

- Είναι αμετάβλητες, δηλαδή δεν μπορούμε να τις τροποιήσουμε.

In [14]:
```
#"Hello world" σε "Hallo world" (δεν πραγματοποιείται)
x = "Hello world!!"
x[2] = 'a'
```

```
MethodError: no method matching setindex!(::String, ::Char, ::Int64)

Stacktrace:
 [1] top-level scope at In[14]:3
 [2] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
```

In [15]:
```
#Το καλύτερο που μπορούμε να κάνουμε είναι να φτιάξουμε μια καινούρια συμβολοσειρά μέσω της x.
x = "Hello world!!"
y = x[1] * "a" * x[3:end]
```

Out[15]: "Hallo world!!"

Υπάρχουν και emojis, όπως 👍 `\:apple: TAB` και 😄 `\:smile: TAB`

Boolean

Εκφράσεις

- Ισότητα = `==`
- Διαφορά = `!=` ή `\ne TAB`
- Συγκρίσεις = `<` , `>` , `<=` ή `\le TAB` , `>=` ή `\ge TAB`

In [16]:
```
5 ≠ 2
```

Out[16]: true

In [17]:
```
"apple" ≥ "newton"
```

Out[17]: false

Λογικοί Τελεστές

- και = `&&`
- ή = `||`
- άρνηση = `!`

In [18]:
```
!(5 == 3) && (5 > 2)
```

Out[18]: true

Επιπρόσθετοι Τελεστές

- x ∈ y = `x \in TAB y` = επιστρέφει true αν ο x χαρακτήρας ανήκει στην συμβολοσειρά y.
- x ∉ y = `x \notin TAB y` = επιστρέφει true αν x χαρακτήρας δεν ανήκει στην συμβολοσειρά y.
- x ≡ y = `x \equiv TAB y` ή `x === y` = επιστρέφει true αν δύο μεταβλητές x και y αναφέρονται στο ίδιο αντικείμενο.

In [19]:
```
'a' ∈ "hola"
```

Out[19]: true

Μπορούμε επίσης να ελέγξουμε το τύπο κάποιου στοιχείου μέσω της συνάρτησης typeof(στοιχείο).

Δηλαδή, `typeof(x)` = επιστρέφει το τύπο του x.

In [20]:
```
#Integer, Ακέραιος αριθμός
typeof(2)
```

Out[20]: Int64

In [21]:
```
#Floating-point number, Αριθμός κινητής υποδιαστολής
typeof(2.5)
```

Out[21]: Float64

In [22]:
```
#"κείμενο" = συμβολοσειρά
typeof("Hello world!!")
```

Out[22]: String

In [23]:
```
#"κείμενο" = συμβολοσειρά
typeof("2.5")
```

Out[23]: String

In [24]:
```
#Boolean τύπος
typeof(true)
```

Out[24]: Bool

In [25]:
```
#Array
typeof([25, 32, "42", [10, 21]])
```

Out[25]: Array{Any,1}

- `x :: Type` = ελέγχει αν το x είναι συγκεκριμένου τύπου. *Χρησιμοποιείται πολύ στις συναρτήσεις και στα structs.*

In [26]:
```
#Έλεγχος του τύπου
(2 + 3) :: Float64
```

```
TypeError: in typeassert, expected Float64, got a value of type Int64

Stacktrace:
 [1] top-level scope at In[26]:2
 [2] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
```

In [27]:
```
(2.0 + 3.2) :: Float64
```

Out[27]: 5.2

Μεταβλητές

- μεταβλητή = έκφραση

In [28]:
```
name1_andras = "Kwstas"
```

Out[28]: "Kwstas"

- Τα ονόματα μεταβλητών μπορούν να αποτελούνται από σχεδόν όλους τους Unicode χαρακτήρες (κεφαλαία και μικρά γράμματα) και από την " _ " (κάτω παύλα), δεν πρέπει να ξεκινούν με κάποιον αριθμό, μπορούν ωστόσο να τον περιέχουν. Προσοχή τα ονόματα των μεταβλητών δεν πρέπει να είναι λέξεις "κλειδιά", δηλαδή true, false, struct, return, etc. λέξεις με κάποια χαρακτηριστική ιδιότητα. *Συνήθως τις ξεχωρίζουμε καθώς αποκτούν χρώμα στον editor*.

In [29]:
```
123name = "Kwstas"
```

```
syntax: "123" is not a valid function argument name around In[29]:1

Stacktrace:
 [1] top-level scope at In[29]:1
 [2] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
```

In [30]:
```
struct = 25
```

```
syntax: unexpected "="
```

- Μπορούν να αποκτήσουν και τις εξής μορφές:
  - x_1 = `x\_1 TAB`
  - y^2 = `y\^2 TAB`

- Αλλαγή έκφραση μεταβλητής:

In [31]:
```
x = 3
y = 5
x += 1 #x = x + 1
y -= x #y = y - x
println("x = ", x)
println("y = ", y)

x = "yes"
println("x = ", x)

#undefined, σημαίνει ότι δεν έχει οριστεί η μεταβλητή.
println(z)
```

```
x = 4
y = 1
x = yes
UndefVarError: z not defined
```

Έλεγχος συνθηκών

Πραγματοποιείται με την εντολή `if`.

- Απλή δομή `if-end`

In [32]:
```
#=
if condition is true
    then do this
end
=#

x = 2
if x > 0
    print("Το $x είναι θετικός αριθμός.")
end
```

```
Το 2 είναι θετικός αριθμός.
```

- Σύνθετες δομές.

1. `if-else-end`

In [33]:
```
#=
if condition is true
    then do this
else
    do this
end
=#
x = 2
y = 5

if x < y
    print("Το $x είναι μικρότερο του $y.")
else
    print("Το $x είναι μεγαλύτερο ίσος του $y.")
end
```

```
Το 2 είναι μικρότερο του 5.
```

1. `if-elseif-else-end`

In [34]:
```
#=
if condition1 is true
    then do this
elseif condition2 is true
    then do this
else
    do this
end
=#

x = 5
y = 5
if x < y
    print("Το $x είναι μικρότερο του $y.")
elseif x > y
    print("Το $x είναι μεγαλύτερος του $y.")
else
    print("Τα x και y είναι ίσα.")
end
```

```
Τα x και y είναι ίσα.
```

1. `if-elseif-elseif-end`

In [35]:
```
#=
if condition1 is true
     then do this
 elseif condition2 is true
     then do this
 elseif condion3 is true
     then do this
end
=#

x = 5
y = 5
if x < y
    print("Το $x είναι μικρότερο του $y.")
elseif x > y
    print("Το $x είναι μεγαλύτερο του $y.")
elseif x == y
    print("Τα x και y είναι ίσα.")
end
```

```
Τα x και y είναι ίσα.
```

1. Nested (Ένθετα)

In [36]:
```
#=
if condition1 is true
    then do this
else
     if condition2 is true
         then do this
     else
         then do this
     end
end
=#

x = 5
y = 2
if x == y
    print("Τα x και y είναι ίσα.")
else
    if x > y
        print("Το $x είναι μεγαλύτερο του $y.")
    else
        print("Το $x είναι μικρότερο του $y.")
    end
end
```

```
Το 5 είναι μεγαλύτερο του 2.
```

In [37]:
```
#Τα παρακάτω παραδείγματα είναι ίδια, έχουν απλώς διαφορετικό τρόπο γραφής.

x = 5

if x > 0
    if x < 8
        println(true)
    end
end


if x > 0 && x < 8
   println(0 < x < 8)
end


if 0 < x < 8
    println("true")
end
```

```
true
true
true
```

- Μπορούμε αντί για if-else-end να γράψουμε `a ? b : c`.

In [38]:
```
#=
Δηλαδή το a ? b : c σημαίνει:

if a
    return b
else
    return c
end
=#

x = 5
y = 3
m = (x>y) ? x : y
# Η εντολή επιστρέφει το μεγαλύτερο από τα x και y.
```

Out[38]:
```
5
```

Δομές επανάληψης

1. `while`

In [39]:
```
#=
while λογική συνθήκη
     <εντολές>
end
=#

#Παράδειγμα αντίστροφης μέτρησης.
x = 3
while x > 0
    println(x)
    x -= 1
end
```

```
3
2
1
```

2. `for`

In [40]:
```
#=
for τιμές μετρητή επανάληψης
      <εντολές>
end
=#

#for μεταβλητή in αρχική τιμή : τελική τιμή
for i in 1:5
    print(i," ")
end

#Μπορούμε αντί για in να χρησιμοποιήσουμε = ή ∈.
print('\n')
for i = 1:5
    print("$i ")
end
print('\n')
for i ∈ 1:5
    print("$i ")
end
println('\n')


#for μεταβλητή in αρχική τιμή : βήμα : τελική τιμή
for i in 1:3:10
    print(i," ")
end
```

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

1 4 7 10
```

- `break` = τερματίζει τις δομές επανάληψης.

In [41]:
```
#Παράδειγμα αντίστροφης μέτρησης.
x = 5
while x > 0
    println(x)
    x -= 1
    if x == 2
        break
    end
end
```

```
5
4
3
```

- `continue` = παραλείπει εντολές και συνεχίζει στην επόμενη τιμή της επανάληψης.

In [42]:
```
#Εκτυπώνει μόνο τους περιττούς αριθμούς
for i in 1:10
    if i % 2 == 0
        continue
    end
    print(i," ")
end
```

```
1 3 5 7 9
```

- `@time` = εκτυπώνει τον χρόνο που χρειάστηκε μια συνάρτηση για να εκτελεστεί, το πλήθος των allocations και το memory allocation.

Συνίσταται να χρησιμοποιείται το @times από το πακέτο BenchmarkTools, για να μην υπολογιστεί ο χρόνος του compilation της συνάρτησης.

Πίνακες

- Διάνυσμα:
  - `a = [2, 3, 4]` ή `a = [2; 3; 4]`
- Πίνακας γραμμή:
  - `a = [2 3 4]`
- Πίνακας στήλη:
  - `a = reshape([2 3 4], 3, 1)`, όπου 3 είναι το πλήθος των γραμμών και 1 το πλήθος των στηλών.
- Πίνακας n x m:
  - `A = [2 3 4; 5 6 7]`
- Ανάστροφος πίνακας, δημιουργείται μέσω του `'`, δηλαδή `A'`.

Δημιουργία κενών πινάκων

- `A = []`

Χαρακτηριστικά / Λειτουργίες

- Είναι μεταβλητοί, δηλαδή μπορούμε να τους τροποποιήσουμε.
  - Για διανύσματα:
    - `όνομα_πίνακα[x]` = εντοπισμός στοιχείου που βρίσκεται στην θέση x του πίνακα, όπου x ακέραιος μεγαλύτερος του 1.
    - `όνομα_πίνακα[x] = y` = τροποποίηση του στοιχείου που βρίσκεται στην θέση x του πίνακα με το στοιχείο y.

- Μπορούμε να επιλέξουμε συγκεκριμένα τμήματα πινάκων.
  - Για διανύσματα:
    - `όνομα_πίνακα[n:m]` = από τον n εώς και τον m δείκτη του διανύσματος.
    - `όνομα_πίνακα[:]` = ολόκληρο το διάνυσμα. _(Ή απλώς γράφουμε το `όνομα_πίνακα`)_
    - `όνομα_πίνακα[n:end]` = από το n στοιχείο μέχρι το τελευταίο του διανύσματος.
    - `όνομα_πίνακα[n:z:m]` = από τον n εώς και τον m δείκτη του διανύσματος με βήμα z.

In [43]:
```
#Array
a = ["5", "deka", 15, [20, 25]]

#Η αρίθμηση δεικτών ξεκινάει από το 1.
#Τροποποίηση του 3ου στοιχείου του διανύσματος.
a[3] = 40
println("1. ", a)

#Μπορούμε να τροποποιήσουμε και περισσότερα στοιχεία ταυτόχρονα ενός διανύσματος.
a[1:3] = [0, 0, 0]
println("2. ", a)
```

```
1. Any["5", "deka", 40, [20, 25]]
2. Any[0, 0, 0, [20, 25]]
```

In [44]:
```
#=
a[:] = ολόκληρο το διάνυσμα.
a[2:4] = από το 2ο εως το 4ο στοιχείο του διανύσματος.
a[3:end] = από το 3ο στοιχείο μέχρι το τελευταίο του διανύσματος.
a[end:-2:3] = από το τελευταίο στοιχείο εώς και το 3ο στοιχείο του διανύσματος με βήμα -2.
=#

a = [1, 3, 5, 4, 2, 6, 7, 8, 9, 10]

println("a[:] = ", a[:])
println("a[2:4] = ", a[2:4])
println("a[3:end] = ",a[3:end])
println("a[end:-2:3] = ",a[end:-2:3])
```

```
a[:] = [1, 3, 5, 4, 2, 6, 7, 8, 9, 10]
a[2:4] = [3, 5, 4]
a[3:end] = [5, 4, 2, 6, 7, 8, 9, 10]
a[end:-2:3] = [10, 8, 6, 4]
```

- Τροποποίηση:
  - Για δυσδιάστατους πίνακες:
    - `όνομα_πίνακα[i, j]` = εντοπισμός στοιχείου που βρίσκεται στην θέση (i, j) του πίνακα, όπου i, j ακέραιοι μεγαλύτεροι του 1.
    - `όνομα_πίνακα[i, j] = y` = τροποποίηση του στοιχείου που βρίσκεται στην θέση (i, j) του πίνακα με το στοιχείο y.

- Επιλογή συγκεκριμένων τμημάτων:
  - Για δυσδιάστατους πίνακες:
    - `όνομα_πίνακα[n:m, k:l]` = από τον n εώς και τον m δείκτη για γραμμές και από τον k εως και τον l δείκτη για στήλες του πίνακα.
    - `όνομα_πίνακα[:,:]` = ολόκληρος ο πίνακας. _(Ή απλώς γράφουμε το `όνομα_πίνακα`)_
    - `όνομα_πίνακα[:]` = ολόκληρος ο πίνακας. *(Σε μορφή διανύσματος, είναι αντίγραφο του πίνακα)*
    - `όνομα_πίνακα[n:end, k:end]` = από τη n μέχρι την τελευταία γραμμή και από την k μέχρι την τελευταία στήλη του πίνακα.

- `όνομα_πίνακα[n:z:m, k:w:l]` = από την n εώς και την m γραμμή του πίνακα με βήμα z, και αντίστοιχα τα k,w,l για τη στήλη.

In [45]:
```
#Δυσδιάστατος πίνακας
A = [2 3; 4 5; 6 1]

#Η αρίθμηση δεικτών ξεκινάει από το 1.
#Τροποποίηση του στοιχείου του πίνακα που βρίσκεται στη θέση (2,1), δεύτερη γραμμή-πρώτη στήλη.
A[2,1] = 40
println("1. A = ", A)

#Μπορούμε να τροποποιήσουμε και περισσότερα στοιχεία ταυτόχρονα ενός πίνακα.
A[1,:] .= 0
println("2. A = ", A)

#Επιπλέον τρόποι:
A[:,2] = [10 32 5]
println("3. A = ", A)

#A[2:3, 1:2] = [0 0;0 0]
#A
```

```
1. A = [2 3; 40 5; 6 1]
2. A = [0 0; 40 5; 6 1]
3. A = [0 10; 40 32; 6 5]
```

In [46]:
```
#=
A[:,:] = ολόκληρος ο πίνακας.
A[2:3, 1:3] = από τη δεύτερη εώς την τρίτη γραμμή του πίνακα και από την πρώτη εώς την τρίτη στήλη.
A[1:end, 2:end] = από την πρώτη γραμμή εώς την τελευταία του πίνακα και από την δεύτερη εώς την τελευταία
A[end:-2:1, end:-1:2] = από την τελευταία γραμμή εώς την πρώτη με βήμα -2 και από την τελευταία στήλη εώς
Προσοχή στα διανύσματα γραμμής και στήλης στο τελευταίο παράδειγμα παίρνουμε τα στοιχεία των διανυσμάτων α
το πρώτο.
=#

A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]

println("A[:,:] = ", A[:,:], " ή A = ", A)
println("A[2:3, 1:3] = ", A[2:3, 1:3])
println("A[1:end, 2:end] = ", A[1:end, 2:end])
println("A[end:-2:1, end:-1:2] = ", A[end:-2:1, end:-1:2])

A
```

```
A[:,:] = [1 3 5; 4 2 6; 7 8 9; 1 2 3] ή A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]
A[2:3, 1:3] = [4 2 6; 7 8 9]
A[1:end, 2:end] = [3 5; 2 6; 8 9; 2 3]
A[end:-2:1, end:-1:2] = [3 2; 6 2]
```

Out[46]:
```
4×3 Array{Int64,2}:
 1  3  5
 4  2  6
 7  8  9
 1  2  3
```

- Μπορούμε να χρησιμοποιήσουμε τον boolean τελεστή ∈, ο οποίος επιστρέφει true αν κάποιο στοιχείο ανήκει στον πίνακα.

In [47]:
```
a = ["5", "deka", 15, [20, 25]]

"deka" ∈ a
```

Out[47]:
```
true
```

In [48]:
```
A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]

15 ∈ A
```

Out[48]:
```
false
```

- Προσπέλαση πινάκων:

In [49]:
```
#Αναφερόμαστε στο στοιχείο του πίνακα.
a = ["5", "deka", 15, [20, 25]]

for item in a
    println(item)
end
```

```
5
deka
15
[20, 25]
```

In [50]:
```
#Αναφερόμαστε στον δείκτη του πίνακα.
a = ["5", "deka", 15, [20, 25]]

for i in eachindex(a)
    println(a[i])
end
```

```
5
deka
15
[20, 25]
```

In [51]:
```
#Ένα ένα τα στοιχεία, όμοιο με τη Matlab.
using Printf
A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]

for i = 1:size(A,1)
    for j = 1:size(A,2)
        @printf("A[%g, %g] = ", i,j)
        println(A[i,j])
    end
end
```

```
A[1, 1] = 1
A[1, 2] = 3
A[1, 3] = 5
A[2, 1] = 4
A[2, 2] = 2
A[2, 3] = 6
A[3, 1] = 7
A[3, 2] = 8
A[3, 3] = 9
A[4, 1] = 1
A[4, 2] = 2
A[4, 3] = 3
```

In [52]:
```
#Ολόκληρη γραμμή
A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]

for row in eachrow(A)
    println(row)
end
```

```
[1, 3, 5]
[4, 2, 6]
[7, 8, 9]
[1, 2, 3]
```

In [53]:
```
#Ολόκληρη στήλη
A = [1 3 5; 4 2 6; 7 8 9; 1 2 3]

for col in eachcol(A)
    println(col)
end
```

```
[1, 4, 7, 1]
[3, 2, 8, 2]
[5, 6, 9, 3]
```

Πράξεις πινάκων

- Ισχύουν οι γνωστές αριθμητικές πράξεις πινάκων με αριθμό και πίνακα, ωστόσο οι εντολές πρέπει να ακολουθούνται από `.` (τελεία), δηλαδή `.+` , `.-` , `./` , `.%` , `.*` , `.^` , ..., αν θέλουμε να εκτελέσουμε πράξεις κατά συντεταγμένες(elementwise).

Χρήσιμες εσωτερικές συναρτήσεις της Julia

Εισόδου

- `readline()` = επιστρέφει σε μορφή συμβολοσειράς ό,τι πληκτρολόγησε ο χρήστης.

In [54]:
```
println("Γράψε μου κάτι"); x = readline()
```

```
Γράψε μου κάτι
stdin> Σήμερα είναι Τετάρτη
```

Out[54]:
```
"Σήμερα είναι Τετάρτη"
```

Εξόδου

- `println(x, y, z, ...)` = εκτυπώνει τα στοιχεία x, y, z και αλλάζει γραμμή (ενσωματωμένο \n).
- `print(x, y, z, ...)` = εκτυπώνει τα στοιχεία x, y, z χωρίς αλλαγές γραμμών.
- `display(x)` = εμφανίζει το στοιχείο x.

Μετατροπή τύπου ενός στοιχείου x

- `convert(τύπος, x)` = επιστρέφει το στοιχείο x στην μορφή τύπου, όπου x είναι αριθμός και τύπος είδος αριθμού (integer, float-point number).
- `parse(τύπος, x)` = επιστρέφει το στοιχείο x στην μορφή τύπου, όπου x είναι συμβολοσειρά ενός αριθμού και τύπος είδος αριθμού (integer, float).

In [55]:
```
convert(Int, 5.0)
```

Out[55]:
```
5
```

In [56]:
```
x = 1/3
display(x)

convert(Float32, x)
```

```
0.3333333333333333
```

Out[56]:
```
0.33333334f0
```

In [57]:
```
parse(Int64, "25")
```

Out[57]:
```
25
```

In [58]:
```
parse(Float64, "25.25")
```

Out[58]:
```
25.25
```

In [59]:
```
parse(Float32, "Spam")
```

```
ArgumentError: cannot parse "Spam" as Float32

Stacktrace:
 [1] _parse_failure(::Type{T} where T, ::String, ::Int64, ::Int64) at .\parse.jl:370 (repeats 2 times)
 [2] #tryparse_internal#364 at .\parse.jl:366 [inlined]
 [3] tryparse_internal at .\parse.jl:364 [inlined]...
```

- `trunc(Int_value, x)` = επιστρέφει σε ακέραια μορφή τον x, όπου x είναι floating-point number και Int_value είναι το είδος ακεραίου.

In [60]:
```
#Η trunc κόβει το δεκαδικό μέρος, δεν στρογγυλοποιεί.
trunc(Int64, 2.955)
```

Out[60]: 2

- `float(x)` = επιστρέφει τον ακέραιο αριθμό x σε μορφή αριθμού κινητής υποδιαστολής. (floating-points number)

In [61]:
```
float(25)
```

Out[61]: 25.0

- `string(x)` = μετατροπή του στοιχείου x σε συμβολοσειρά.

In [62]:
```
string(25.25)
```

Out[62]: "25.25"

In [63]:
```
string("Ο αριθμός ", 10, " διαιρείται με το 2")
```

Out[63]: "Ο αριθμός 10 διαιρείται με το 2"

- `bitstring(x)` = επιστρέφει τη δυαδική αποθήκευση του αριθμού x σε λέξη 64 bits.

In [64]:
```
println(bitstring(1.0))
println(bitstring(1.0+eps(Float64)))
println(bitstring(2.0))
```

```
0011111111110000000000000000000000000000000000000000000000000000
0011111111110000000000000000000000000000000000000000000000000001
0100000000000000000000000000000000000000000000000000000000000000
```

Μαθηματικών

- `sqrt(x)` = επιστρέφει $\sqrt{x}$.
- `log(x)` = επιστρέφει $ln(x)$.
- `log10(x)` = επιστρέφει $log(x)$.
- `exp(x)` = επιστρέφει το e^x
- `sin(x)` = επιστρέφει το ημίτονο του x.
- `cos(x)` = επιστρέφει το συνημίτονο του x.
- `tan(x)` = επιστρέφει την εφαπτομένη του x.
- `abs(x)` = επιστρέφει την απόλυτη τιμή του x.
- `factorial(x)` = επιστρέφει το παραγοντικό του ακεραίου x.
- `div(x, y)` = επιστρέφει το πηλίκο των αριθμών x, y.
- `rem(x, y)` = επιστρέφει το υπόλοιπο ων αριθμών x, y.
- `divrem(x, y)` = επιστρέφει σε μορφή πλειάδας (πηλίκο, υπόλοιπο) των αριθμών x, y.
- `rand()` = επιστρέφει έναν τυχαίο αριθμό από το διάστημα [0, 1).
- `rand(n:m)` = επιστρέφει έναν τυχαίο ακέραιο αριθμό από το διάστημα [n, m]

Συμβολοσειρών

- `uppercase(x)` = επιστρέφει με κεφαλαία γράμματα την συμβολοσειρά x.
- `findfirst(x, string)` = επιστρέφει την πρώτη θέση-εις που εμφανίζεται η συμβολοσειρά x.
- `findnext(x, string, number)` = όμοια με την findfirst μόνο που ξεκινάει τον έλεγχο από την θέση number.
- `collect(x)` = μετατροπή συμβολοσειράς x σε πίνακα, όπου κάθε στοιχείο αντιστοιχεί σε έναν χαρακτήρα της x.
- `split(x)` = μετατροπή συμβολοσειράς x σε πίνακα, όπου κάθε στοιχείο αντιστοιχεί σε μία λέξη της x.
- `split(x, k)` = μετατροπή συμβολοσειράς x σε πίνακα, όπου κάθε στοιχείο προέκυψε απο την αποκοπή του k χαρακτήρα από την x.

Πινάκων

- `push!(a, x)` = εισαγωγή του στοιχείου x στο τέλος του πίνακα a.
- `pushfirst!(a, x)` = εισαγωγή του στοιχείου x στην αρχή του πίνακα a.
- `append!(a, b)` = ένωση δύο πινάκων array a και b, όπου τα στοιχεία του b εισάγονται στο τέλος του πίνακα a.
- `insert!(a, index, x)` = εισαγωγή του στοιχείου x στην θέση index του πίνακα a.
- `sort!(a)` = ταξινομούνται κατά αύξουσα σειρά τα στοιχεία του πίνακα a.
- `sort(a)` = επιστρέφει αντίγραφο του πίνακα a με τα στοιχεία του ταξινομημένα κατά αύξουσα σειρά.
- `sum(a)` = το άθροισμα των στοιχείων του a.
- `splice!(a, index)` = διαγράφει το στοιχείο που βρίσκεται στη θέση index του πίνακα a και το επιστρέφει.
- `pop!(a)` = διαγράφει το τελευταίο στοιχείο του πίνακα a και το επιστρέφει.
- `popfirst!(a)` = διαγράφει το πρώτο στοιχείο του πίνακα a και το επιστρέφει.
- `deleteat!(a, index)` = διαγράφει το στοιχείο που βρίσκεται στη θέση index του πίνακα a. (δεν το επιστρέφει)
- `join(a, k)` = ένωση των στοιχείων του πίνακα a σε μια συμβολοσειρά όπου τα στοιχεία ενώνονται με τον χαρακτήρα k.
- `rand(n, m)` = επιστρέφει έναν πίνακα n x m με στοιχεία που βρίσκονται στο διάστημα [0,1).
- `rand(a:b, n, m)` = επιστρέφει έναν πίνακα n x m με στοιχεία που βρίσκονται στο διάστημα [a,b].
- `rand(n, m ,z)` = επιστρέφει έναν πίνακα n x m x z με στοιχεία που βρίσκονται στο διάστημα [0,1).
- `zeros(n, m)` = επιστρέφει έναν πίνακα n x m με μηδενικά στοιχεία.
- `ones(n, m)` = επιστρέφει έναν πίνακα n x m με στοιχεία άσσους.
- `size(A,1)` = επιστρέφει το πλήθος των γραμμών ενός πίνακα.
- `size(A,2)` = επιστρέφει το πλήθος των στηλών ενός πίνακα.
- `(m, n) = size(A)` ή `m, n = size(A)` = επιστρέφει το πλήθος των γραμμών στη μεταβλητή m και των στηλών στη μεταβλητή n.

Λεξικών

- `lenght(dict)` = επιστρέφει το πλήθος των στοιχείων (key-value) του λεξικού.
- `keys(dict)` = επιστρέφει σε λίστα τα κλειδιά του λεξικού.
- `values(dict)` = επιστρέφει σε λίστα τα values του λεξικού.
- `pop!(dict, key)` = διαγράφει το key από το λεξικό και επιστρέφει το value του.

Πλειάδων

Όταν θέλουμε να εισάγουμε πλειάδα σε συνάρτηση που δέχεται περισσότερες από μια εισόδους, γράφουμε το όνομα της μεταβλητής και αμέσως μετα τρεις τελείες. `(t...)`

Παράδειγμα: Η divrem δέχεται δύο αριθμούς. Αν t = (x, y) πλειάδα, τότε τρέχουμε την divrem ως: `divrem(t...)`

- `zip(x1, x2, ...)` = δέχεται αλληλουχίες και επιστρέφει μια συλλογή από πλειάδες, όπου κάθε μία περιέχει ένα στοιχείο από κάθε αλληλουχία.

Structs

- `fieldnames(όνομα_struct)` = επιστρέφει τα πεδία του struct.
- `isdefined(object, :πεδίο)` = επιστρέφει true/false σε περίπτωση που υπάρχει/δεν υπάρχει το πεδίο στο object του struct.
- `object isa όνομα_struct` = επιστρέφει true/false σε περίπτωση που το object είναι/δεν είναι instance του struct.

Επιπρόσθετες συναρτήσεις

- `minimum(x)` = επιστρέφει το μικρότερο στοιχείο μιας αλληλουχίας x.
- `maximum(x)` = επιστρέφει το μεγαλύτερο στοιχείο μιας αλληλουχίας x.
- `reverse(x)` = επιστρέφει σε ανεστραμμένη μορφή μια αλληλουχία x.
- `deepcopy(x)` = αντιγραφή του x.

Συναρτήσεις με/χωρίς `!` , `.`

- Οι συναρτήσεις που ακολουθούνται από `!` μεταλλάσουν (mutating functions) το περιεχόμενο τους, ενώ όσες δεν ακολουθούνται από `!`, το περιεχόμενο τους παραμένει αμετάβλητο. (non-mutating functions)

In [65]:
```
#Παράδειγμα mutating, non-mutating functions.
v = [3, 4, 1]

#Non-mutating
println("sorted_v = ", sort(v))
println("v = ", v, "\n")

#Mutating
println("sorted!_v = ", sort!(v))
println("v = ", v)
```

```
sorted_v = [1, 3, 4]
v = [3, 4, 1]

sorted!_v = [1, 3, 4]
v = [1, 3, 4]
```

- Οι συναρτήσεις που ακολουθούνται από `.` "εξετάζουν" σε μέρη/κομμάτια το object, ενώ όσες δεν ακολουθούνται από `.` "εξετάζουν" το object ως ενιαίο.

In [66]:
```
#Παράδειγμα
f1(A) = A^2

B = [2 3; 5 6]

println("f1(B) = ", f1(B))
println("f1.(B) = ", f1.(B))
```

```
f1(B) = [19 24; 40 51]
f1.(B) = [4 9; 25 36]
```

Υλοποίηση Συναρτήσεων

Χρησιμοποιούμε την εντολή `function`

- Παράδειγμα συνάρτησης χωρίς είσοδο.

In [67]:
```
#=
Παράδειγμα συνάρτησης που εκτυπώνει αντικείμενα δωματίου.
Δωμάτιο = Υπνοδωμάτιο
=#

function room_items()
    print("Κρεβάτι ", "Γραφείο ")
end
```

Out[67]:
```
room_items (generic function with 1 method)
```

In [68]:
```
#=
Παράδειγμα συνάρτησης που εκτυπώνει επιπλέον αντικείμενα δωματίου.
Εμφωλευμένες συναρτήσεις.
=#

function extra_room_items()
    room_items()
    print("Καρέκλα")
end
```

Out[68]:
```
extra_room_items (generic function with 1 method)
```

In [69]:
```
#Εν τέλη έχουμε:

function room_items()
    print("Κρεβάτι ", "Γραφείο ")
end

function extra_room_items()
    room_items()
    print("Καρέκλα")
end

extra_room_items()
```

```
Κρεβάτι Γραφείο Καρέκλα
```

- Παράδειγμα συνάρτησης με είσοδο.

In [70]:
```
#=
Παράδειγμα συνάρτησης που εκτυπώνει αντικείμενα δωματίου.
Δωμάτιο = όποιο επιθυμεί ο χρήστης.
Αντικείμενα = όποια επιθυμεί ο χρήστης.
=#

function room_items(room, item1, item2)
    print("Τα αντικείμενα του δωματίου ", room, " είναι ", item1, " και ", item2)
end

room_items("σαλόνι", "τηλεόραση", "καναπές")
```

```
Τα αντικείμενα του δωματίου σαλόνι είναι τηλεόραση και καναπές
```

In [71]:
```
#Παράδειγμα συνάρτησης με αριθμητικές πράξεις.

function operations(x,y,z)
    a = (x + y)*z
end

operations(112,2.34, [1 2; 4 5])
```

Out[71]:
```
2×2 Array{Float64,2}:
 114.34  228.68
 457.36  571.7
```

- Μπορούμε επίσης να υλοποιήσουμε συναρτήσεις μιας γραμμής.

In [72]:
```
#Συνάρτηση που υπολογίζει το γινόμενο δύο αριθμών.
f(a, b) = a*b

#=
Είναι ισοδύναμο με:
function f(a, b)
    a*b
end
=#
```

Out[72]:
```
f (generic function with 1 method)
```

In [73]:
```
f(2, 3)
```

Out[73]:
```
6
```

In [74]:
```
room_items(room, item1, item2) = print("Τα αντικείμενα του δωματίου ", room, " είναι ", item1, " και ", it
```

Out[74]:
```
room_items (generic function with 2 methods)
```

In [75]:
```
room_items("σαλόνι", "τηλεόραση", "καναπές")
```

```
Τα αντικείμενα του δωματίου σαλόνι είναι τηλεόραση και καναπές
```

Για να **επιστρέψουμε** values από τις συναρτήσεις χρησιμοποιούμε το `return`.

In [76]:
```
#Παράδειγμα συνάρτησης που επιστρέφει το άθροισμα ή το γινόμενο δύο αριθμών.
function calculate_items(item1, item2)
    if item1 > item2
        return item1+item2
    else
        return item1*item2
    end
end

#Επιστρέφουν το αποτέλεσμα.
#calculate_items(8,5)
calculate_items(3,5)
```

Out[76]:
```
15
```

- Για να εισάγουμε στις συναρτήσεις **global** μεταβλητές, αρκεί να περιέχεται η εντολή `global όνομα_μεταβλητής` μέσα στον κώδικα της συνάρτησης. (Αντίστοιχα υπάρχει και η εντολή `local`)

Λεξικά

Δημιουργία Λεξικού

- `x = Dict()` = κενό λεξικό
- `x = Dict(key => value, ...)` = λεξικό με στοιχεία

In [77]:
```
#Δημιουργία μη κενού λεξικού
x = Dict(1 => "one", 2 => "two", 4 => "four")

#=
Int64: Τα κλειδιά είναι ακέραιοι
String: Τα values συμβολοσειρές
Any: Παραπάνω από ένα είδος
=#
```

Out[77]:
```
Dict{Int64,String} with 3 entries:
  4 => "four"
  2 => "two"
  1 => "one"
```

Προσθήκη στοιχείων.

- `x[key] = value`
- `x = Dict(key => value)`

In [78]:
```
#Δημιουργία κενού λεξικού (έχει any) και στη συνέχεια προσθήκη ενός στοιχείου.
x = Dict()
x[1] = "one"
x
```

Out[78]:
```
Dict{Any,Any} with 1 entry:
  1 => "one"
```

Αναφορά σε στοιχείο

- `x[key]`

In [79]:
```
x = Dict(1 => "one", 2 => "two", 4 => "four")
x[2]
```

Out[79]:
```
"two"
```

- Μπορούμε να χρησιμοποιήσουμε τον boolean τελεστή ∈, ο οποίος επιστρέφει true αν κάποιο key ή/και value υπάρχει στο λεξικό.

In [80]:
```
x = Dict(1 => "one", 2 => "two", 4 => "four")

#Κλειδί
k = keys(x)
println("1) ", 2 ∈ k)

#Value
v = values(x)
println("2) ", "two" ∈ v)

#Στοιχείο
println("3) ", (2=>"two") ∈ x)
```

```
1) true
2) true
3) true
```

In [81]:
```julia
#=
Παράδειγμα εκτύπωσης value των λεξικών.
(k,v) = πλειάδα, όπου k το key και v το value.
=#

x = Dict(1 => "one", 2 => "two", 4 => "four")

for (k,v) in x
    println(x[k])
end
```

```
four
two
one
```

Πλειάδες

Δημιουργία Πλειάδων

- Με παρένθεση: `t = (a, b, c, ...)`
- Χωρίς παρένθεση: `t = a, b, c, ...`
- Με χρήση της συνάρτησης `tuple()` :
  - `t = tuple()` = κενή πλειάδα
  - `t = tuple(a, b ,c, ...)`

όπου a, b, c,... είναι οποιαδήποτε στοιχεία, π.χ. αριθμοί, χαρακτήρες, συμβολοσειρές, πίνακες,...

- Η πλειάδα με ένα στοιχείο πρέπει να είναι της μορφής: `t =(a, )`

In [82]:
```julia
#Παραδείγματα πλειάδων.
t1 = 2, "asdf", 4_324.312 ,[2 3; 2 2]
t2 = (2, "asdf", 4_324.312 ,[2 3; 2 2])
t3 = tuple(2, "asdf", 4_324.312 ,[2 3; 2 2])

println("t1 = ", t1)
println("t2 = ", t2)
println("t3 = ", t3)

#Πλειάδα με ένα στοιχείο.
t4 = ('a',)
println("t4 = ", t4,", όπου t4 είναι ", typeof(t4))

#Δεν είναι πλειάδα.
t5 = ('a')
println("t5 = ", t5,", όπου t5 είναι ", typeof(t5))
```

```
t1 = (2, "asdf", 4324.312, [2 3; 2 2])
t2 = (2, "asdf", 4324.312, [2 3; 2 2])
t3 = (2, "asdf", 4324.312, [2 3; 2 2])
t4 = ('a',), όπου t4 είναι Tuple{Char}
t5 = a, όπου t5 είναι Char
```

Χαρακτηριστικά / Λειτουργίες

- Είναι αμετάβλητοι, δηλαδή δεν μπορούμε να τις τροποποιήσουμε.
- Μπορούμε να επιλέξουμε συγκεκρίμενα τμήματα πλειάδων, όπως και στους πίνακες:
  - `όνομα_πλειάδας[index]` = αναφορά σε ένα στοιχείο της πλειάδας που βρίσκεται στην θέση index της πλειάδας, όπου index ακέραιος μεγαλύτερος του 1.
  - `όνομα_πλειάδας[n:m]` = αναφορά στο τμήμα της πλειάδας από τη θέση n εώς m.
- Αναγνωρίζουν τους τελεστές σύγκρισης.

Structs

Υλοποίηση struct

Χρησιμοποιούμε την εντολή `struct`

In [83]:
```
#=
struct όνομα
    body -> πεδία_του_struct
end
=#

#Παράδειγμα struct σημείο του τρισδιάστατου χώρου.
struct Point
    x
    y
    z
end
```

Αναφορά στα values του struct

- `όνομα_struct.value`

In [84]:
```
#Παράδειγμα με το struct Point.

println("x = ", Point(1,2,3).x)

y = Point(1,2,3).y
println("y = ",y)

point = Point(1,2,3)
z = point.z
println("z = ",z)
```

```
x = 1
y = 2
z = 3
```

Σημαντικό: Τα structs (`struct-end`) είναι αμετάβλητα, immutable.

In [85]:
```
#Παράδειγμα αλλαγής τιμής του y.
point = Point(1,2,3)
point.y = 4
```

```
setfield! immutable struct of type Point cannot be changed

Stacktrace: [1] setproperty!(::Point, ::Symbol, ::Int64) at .\Base.jl:34...
```

Μεταβλητά structs

Μπορούμε ωστόσο να δημιουργήσουμε μεταβλητά structs με την εντολή `mutable struct` .

In [86]:
```
#=
mutable struct όνομα
    body -> πεδία_του_mutable_struct
end
=#

#Παράδειγμα mutable struct σημείο του τρισδιάστατου χώρου.
mutable struct MutablePoint
    x
    y
    z
end
```

In [87]:
```
mp = MutablePoint(0, 0, 0)
mp.x = 1
mp.y = 2
mp.z = 3
println(mp)
```

```
MutablePoint(1, 2, 3)
```

Αρχεία

- `open("όνομα_αρχείου")` = άνοιγμα αρχείου
- `open("όνομα_αρχείου", "w")` = άνοιγμα αρχείου in write mode.
  - `write(file, string)` = επιστρέφει το πλήθος των χαρακτήρων του string, που εισάγαμε/"γράψαμε" στο αρχείο file.
- `readline("όνομα_αρχείου")` = ανάγνωση μιας γραμμής αρχείου
- `close("όνομα_αρχείου")` = κλείσιμο αρχείου
- `pwd()` = επιστρέφει το όνομα του current working directory.
- `abspath("όνομα_αρχείου")` = επιστρέφει το absolut path του αρχείου.
- `ispath("όνομα_αρχείου")` = επιστρέφει true/false, σε περίπτωση που υπάρχει/δεν υπάρχει το αρχείο.
- `isdir("something")` = επιστρέφει true/false, σε περίπτωση που το something είναι/δεν είναι directory.
- `isfile("something")` = επιστρέφει true/false, σε περίπτωση που το something είναι/δεν είναι αρχείο.
- `readdir(directory)` = επιστρέφει συμβολοσειρά από αρχεία (και άλλα directories) που υπάρχουν στο δοσμένο directory.

Packages

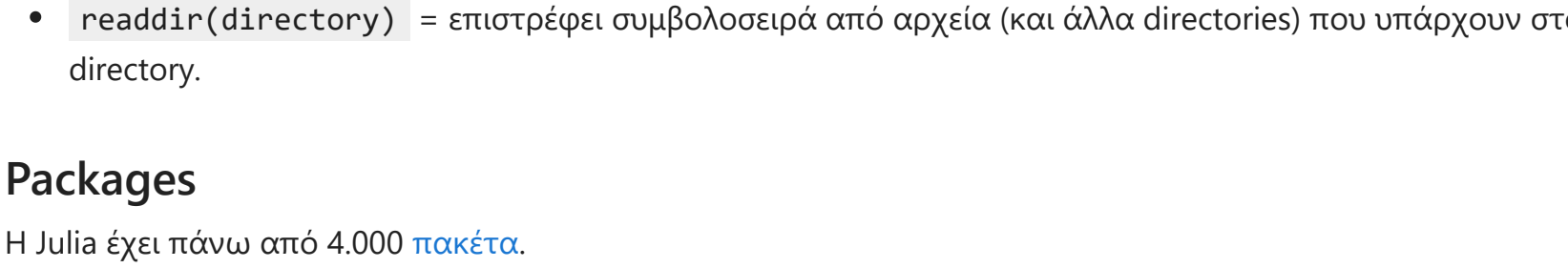
Η Julia έχει πάνω από 4.000 πακέτα.

- Προσθήκη πακέτων: `Pkg.add("πακέτο")`
- Χρήση πακέτου: `using πακέτο`
- Διαγραφή πακέτου: `Pkg.rm("πακέτο")`
- Αναβάθμιση πακέτου: `Pkg.update("πακέτο")`

In [88]:

```
#=
import Pkg
Pkg.add("Colors")
=#

using Colors

#Δημιουργία παλέτας με 100 διαφορετικά χρώματα.
palette = distinguishable_colors(100)
```

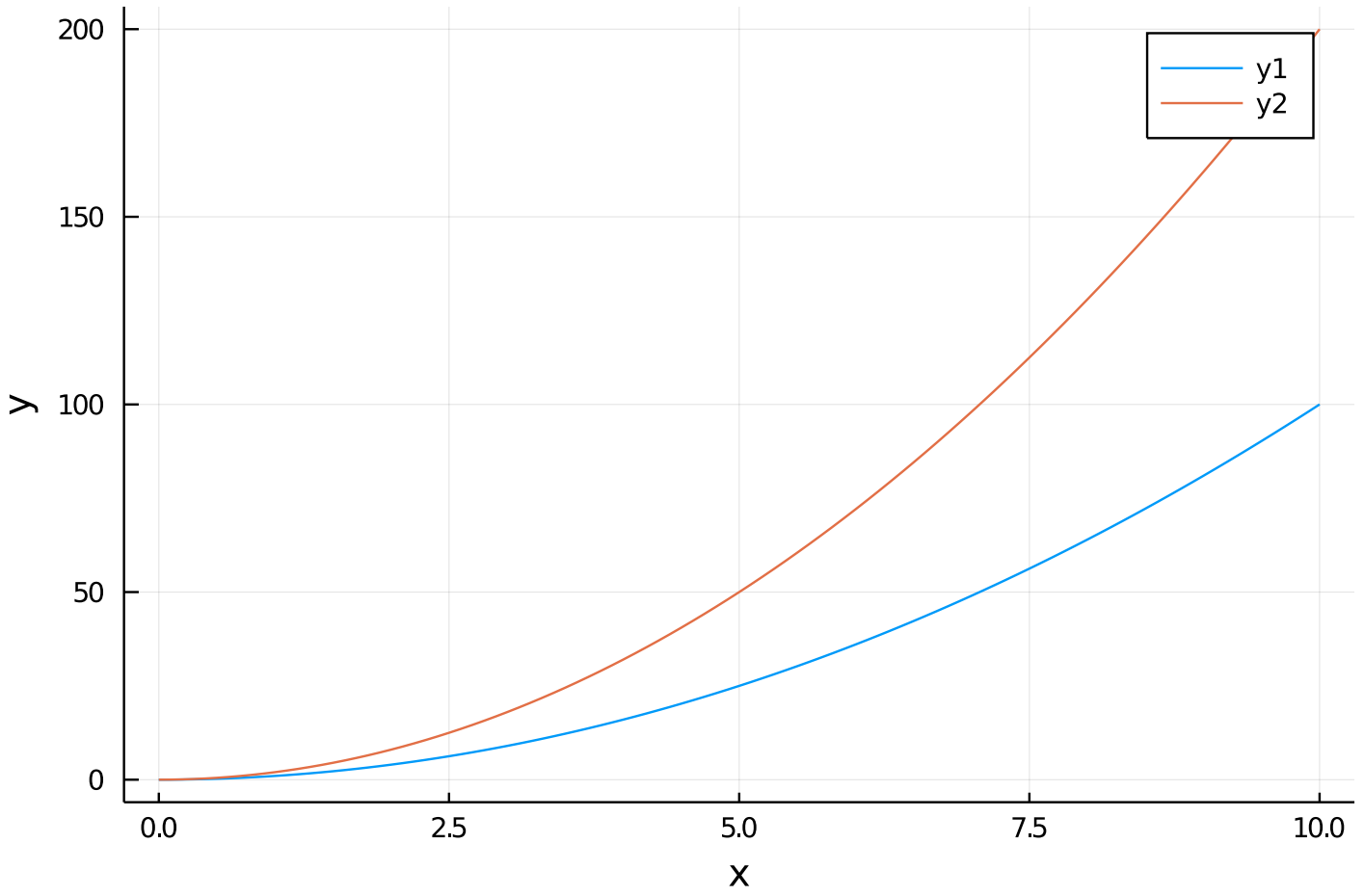
Out[88]:



In [89]:

```
#=
import Pkg
Pkg.add("Plots")
=#

using Plots

x = 0:0.1:10
y = x.^2

plot(x, y, xlabel ="x", ylabel = "y", label = "y1")
plot!(x, 2*y, xlabel ="x", ylabel = "y", label = "y2")

#Αν γράφαμε plot χωρίς !, τότε δεν θα εμφανιζόταν το y1.
#Χρησιμοποιούμε gr() για σχεδιασμό.
```

Out[89]:

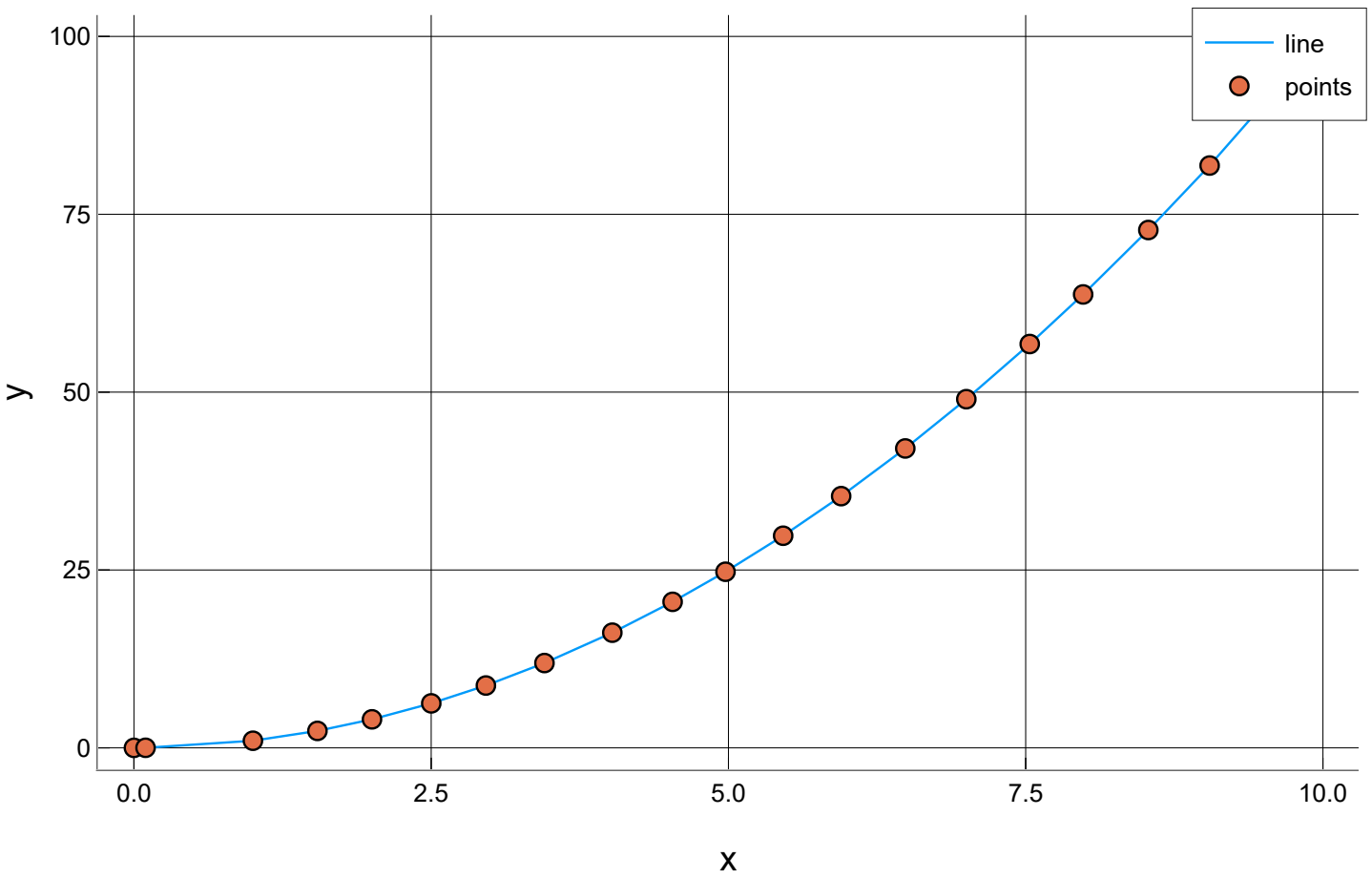


In [90]:

```
#import Pkg; Pkg.add("PlotlyJS")

plotlyjs()
plot(x -> x^2, 0, 10, xlabel ="x", ylabel = "y", label = "line")
scatter!(x -> x^2, 0, 10, xlabel ="x", ylabel = "y", label = "points")
```

Unable to load WebIO. Please make sure WebIO works for your Jupyter client. For troubleshooting, please see the WebIO/IJulia documentation.

Out[90]:



In [91]:

```
#subplot
x = 0:0.5:10

y1 = plot(x, x, label = "y1")
y2 = plot(x, x.^2, label = "y2")
y3 = plot(x, x.^3, label = "y3")
y4 = plot(x, x.^4, label = "y4")

plot(y1,y2,y3,y4)
```

Out[91]: