

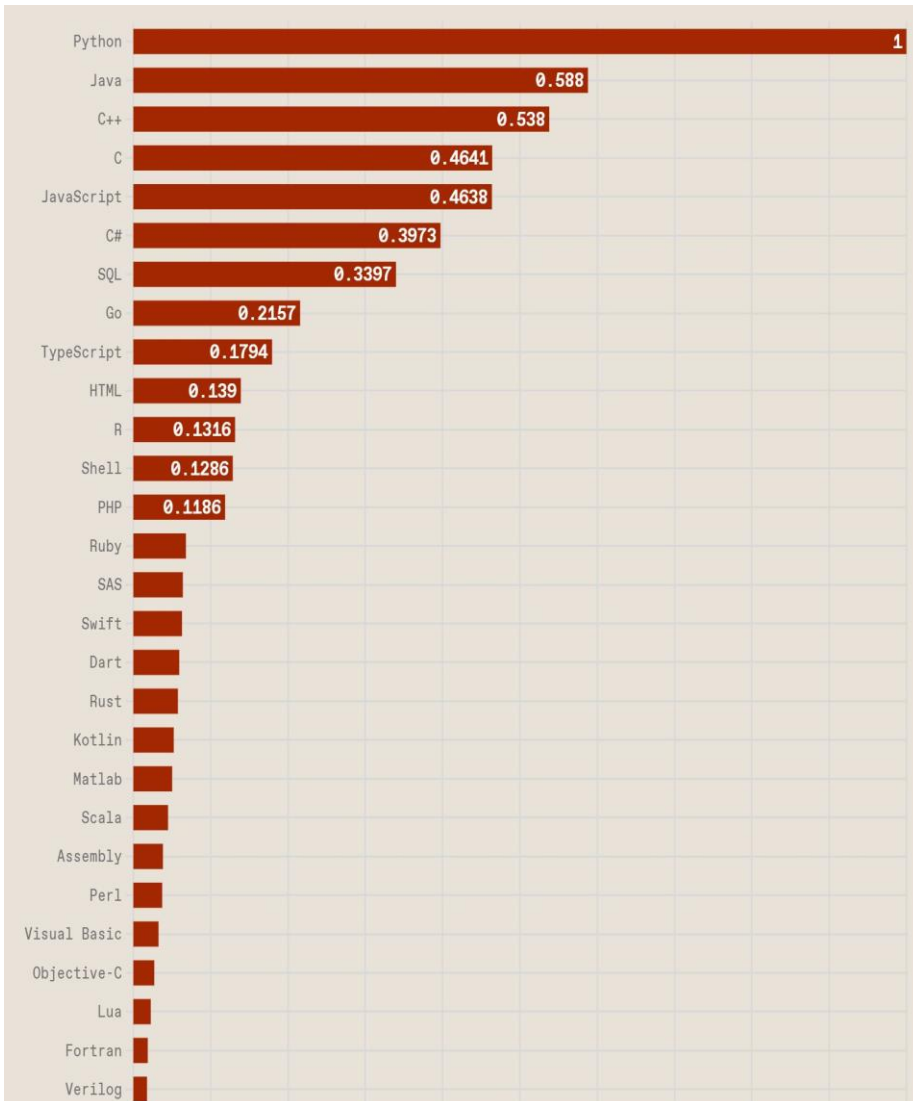
# Κεφάλαιο 1: Εισαγωγή

*Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών*

Γιάννης Γαροφαλάκης, Σπύρος Σιούτας,  
Παναγιώτης Χατζηδούκας

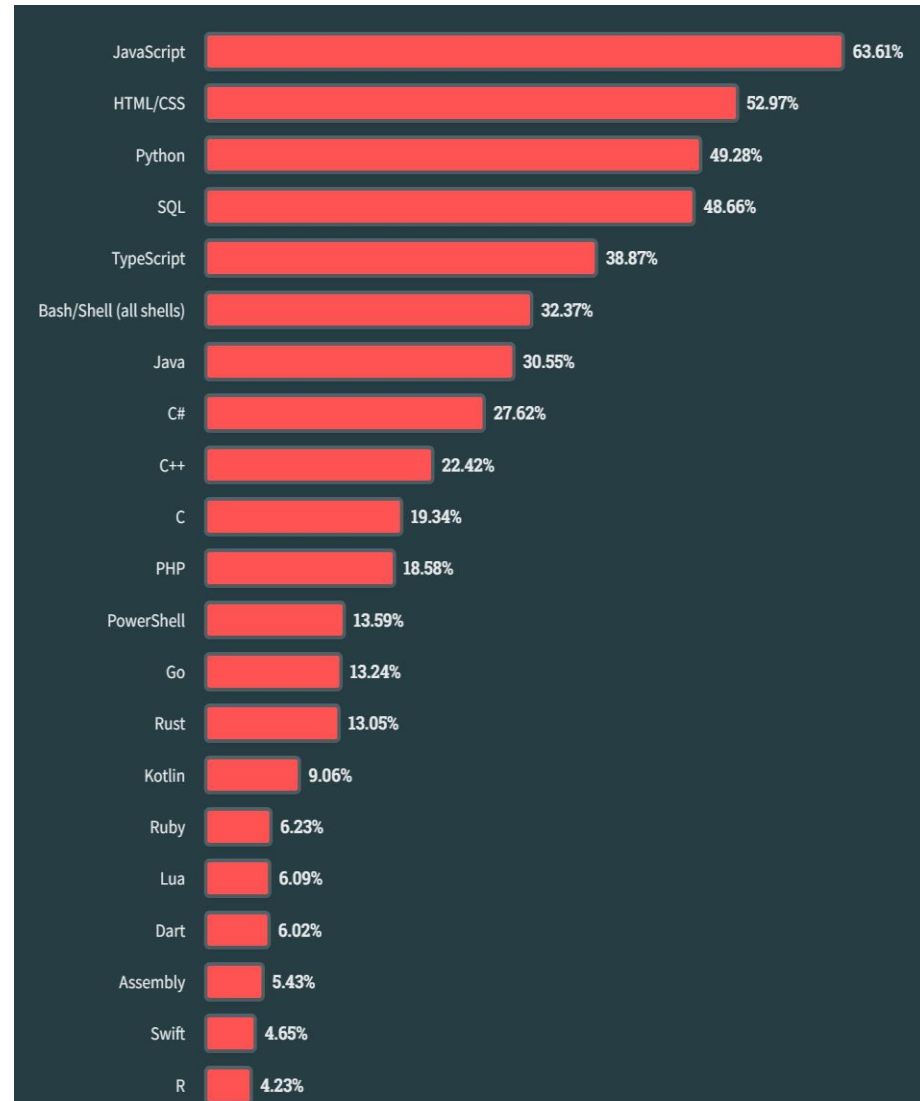
# IEEE Spectrum Top Programming Languages 2023

<https://spectrum.ieee.org/the-top-programming-languages-2023>



# StackOverflow Developer Survey 2023

<https://survey.stackoverflow.co/2023/#overview>



IEEE Computer Society - Gaurav Belani

# “Programming Languages You Should Learn in 2020” (and Shannon Flynn 2022)

- Η συνεχώς αυξανόμενη λίστα Γλωσσών Προγραμματισμού (ΓΠ) και σχετικών πρωτοκόλλων, κάνει δύσκολη την επιλογή μιας ΓΠ κατάλληλης για τη δουλειά του προγραμματιστή.
- Πάντως, κάθε προγραμματιστής που σέβεται τον εαυτό του, θα πρέπει να έχει γνώση από Γλώσσες Προγραμματισμού για:
  - System programming (π.χ. **C**, **Go**, **C++**)
  - Object-oriented programming (π.χ. **Java**, **Python**)
  - Functional programming (π.χ. **Scala**, **Python**)
  - Scripting (π.χ. **JavaScript**, **Python**)

# “Programming Languages You Should Learn in 2020” (and Shannon Flynn 2022) – 2

## ■ Hot Γλώσσες Προγραμματισμού:

1. **Python**
2. **Kotlin** (για Android apps, interop with Java)
3. **Java**
4. **JavaScript / NodeJS**
5. **TypeScript** (υπερσύνολο της JavaScript, από MS)
6. **Go (-lang)** (system level, για cloud συστήματα, Google)
7. **Swift** (για iOS και macOS apps, interop with C, C++)

## ■ Επιπλέον:

8. **R** (για Statistics, Data Analysis, Machine Learning)
9. **C#** (.NET, από MS)
10. **PHP** (πολύ χρήσιμη ακόμα...)
11. **C/C++** (προφανώς...)

# Εισαγωγή (1)

- Γιατί, όμως, υπάρχουν τόσες **πολλές** Γλώσσες Προγραμματισμού (ΓΠ);
  - *Εξέλιξη* – έχουμε μάθει καλύτερους τρόπους να κάνουμε πράγματα με το πέρασμα του χρόνου
  - *Κοινωνικό-οικονομικοί Παράγοντες*: ιδιωτικά ενδιαφέροντα, εμπορικό πλεονέκτημα
  - *Προσανατολισμός σε ειδικούς σκοπούς* (π.χ. εφαρμογές στο Web)
  - *Προσανατολισμός σε ειδικό hardware*
  - *Διάφορες ιδέες για το τι είναι ευχάριστο να χρησιμοποιείται*

# Εισαγωγή (2)

- Τι κάνει μια γλώσσα προγραμματισμού **επιτυχημένη**;
  - Ευκολία στη *μάθηση* (BASIC, Pascal, Python)
  - Ευκολία στο να *εκφραστούν* πράγματα, ευκολία στη *χρήση* όταν τη μάθει κανείς (C, Common Lisp, APL, Algol-68, PHP, Python)
  - Ευκολία στην *ανάπτυξη* (BASIC)
  - Δυνατότητα να μεταγλωττιστεί σε *πολύ καλό* (γρήγορο/μικρό) *κώδικα* (Fortran)
  - Να έχει ένα δυνατό *υποστηρικτή* (COBOL, PL/1, Ada, Visual Basic, πλέον Google κ.α.)
  - Μεγάλη διασπορά με *ελάχιστο κόστος* (Pascal, Java, Python)

# Εισαγωγή (3)

- **Γιατί έχουμε** γλώσσες προγραμματισμού; Τι κάνει μια ΓΠ;
  - Τρόπος σκέψης – τρόπος έκφρασης αλγορίθμων
  - Γλώσσες από τη σκοπιά που σκέφτεται ο *χρήστης*
  - *Αφαίρεση* εικονικής μηχανής – τρόπος καθορισμού του *τι θέλουμε*
  - Τρόπος να κάνουμε το hardware να κάνει κάτι *χωρίς* να μπλεχτούμε με τα *bits*
  - Γλώσσες από τη σκοπιά του *κατασκευαστή*

# Γιατί να μάθει κανείς Αρχές Γλωσσών Προγραμματισμού; (1)

- Βοήθεια για να επιλεχθεί μια γλώσσα. Π.χ.:
  - **C vs C++** για προγραμματισμό συστημάτων
  - **Fortran vs Ada** για αριθμητικούς υπολογισμούς
  - **Common Lisp vs Scheme vs ML** για διαχείριση συμβολικών δεδομένων
  - **Java vs C** για δικτυωμένα προγράμματα υπολογιστών
  - **NodeJS vs PHP** για server-side scripting



# Γιατί να μάθει κανείς Αρχές

## Γλωσσών Προγραμματισμού; (2)

- Μερικές γλώσσες είναι παρόμοιες: είναι εύκολο να μάθεις μια γλώσσα αν ξέρεις τη γλώσσα στην οποία βασίστηκε.
  - Οι έννοιες έχουν ακόμη μεγαλύτερη ομοιότητα: αν σκεφτεί κανείς τις έννοιες της επανάληψης, αναδρομής, αφαίρεσης, θα βρει πιο εύκολο να αφομοιώσει τη σύνταξη και τις σημασιολογικές λεπτομέρειες μιας νέας γλώσσας από το να προσπαθήσει κάτι από την αρχή. Σε αναλογία με τις ανθρώπινες γλώσσες: μεγάλος έλεγχος της γραμματικής κάνει πιο εύκολη την εκμάθηση μιας νέας γλώσσας (τουλάχιστον ινδοευρωπαϊκή)

Γιατί να μάθει κανείς Αρχές

Γλωσσών Προγραμματισμού; (3)

- Βοήθεια για να κάνει κανείς καλύτερη χρήση οποιασδήποτε γλώσσας προγραμματισμού
  - Να κατανοηθούν τα δυσνόητα χαρακτηριστικά
    - Π.χ. στη C, βοηθάει να κατανοηθούν ενώσεις, πίνακες και δείκτες, η ξεχωριστή μεταγλώττιση
  - Εύρεση για το πως μπορούν να γίνουν πράγματα σε μια γλώσσα όταν αυτή δεν τα υποστηρίζει ρητά:
    - Έλλειψη κατάλληλων δομών ελέγχου και αναδρομής στη Fortran
    - Έλλειψη ονομασμένων σταθερών και απαριθμήσεων στη Fortran
    - Χρήση μεταβλητών που αρχικοποιήθηκαν μια φορά, και μετά ποτέ δεν αλλάζουν

# Ορισμοί (1)

- Γλώσσα Προγραμματισμού (ΓΠ):

*Συστηματική Σημειογραφία με την οποία περιγράφουμε  
υπολογιστικές διεργασίες*



Σύνολο βημάτων που μπορεί να εκτελέσει μια μηχανή για να λύσει ένα πρόβλημα

Η περιγραφή της λύσης ενός προβλήματος στον Η/Υ προϋποθέτει ένα σύνολο **εντολών** που καταλαβαίνει και εκτελεί ο Η/Υ

Ο **Αλγόριθμος** περιγράφει μεθόδους και διαδικασίες για τον τρόπο χειρισμού των δεδομένων εισόδου, με στόχο την ικανοποίηση των επιθυμητών στόχων

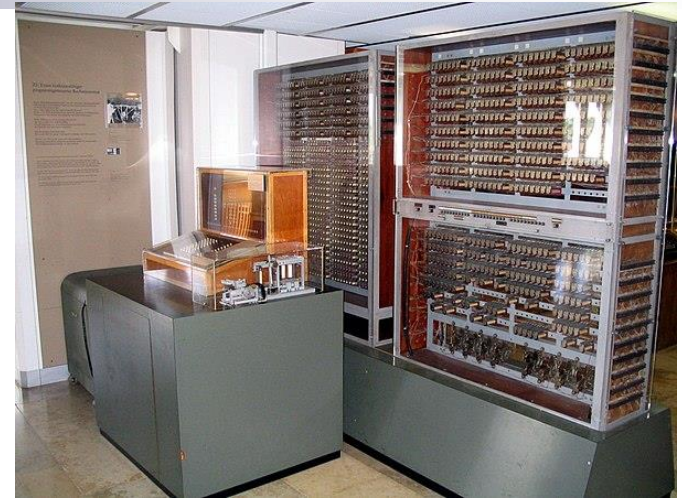
# Ορισμοί (2)

Εντολή για την ανάθεση της τιμής 2 σε μεταβλητή:  
(εξέλιξη)

1. Γλώσσα μηχανής (Intel 8x86, PCs)  
`C7 06 0000 0002` (μετακίνηση του 2 στη δ/νση 0000, 16δικό)
2. Assembly (εξαρτώμενη από μηχανή)  
`MOV X, 2`
3. Γλώσσα Προγραμματισμού Υψηλού Επιπέδου  
`X = 2`

# Ιστορικά (1)

Z3



## ■ Δεκαετίες 1930 & 1940

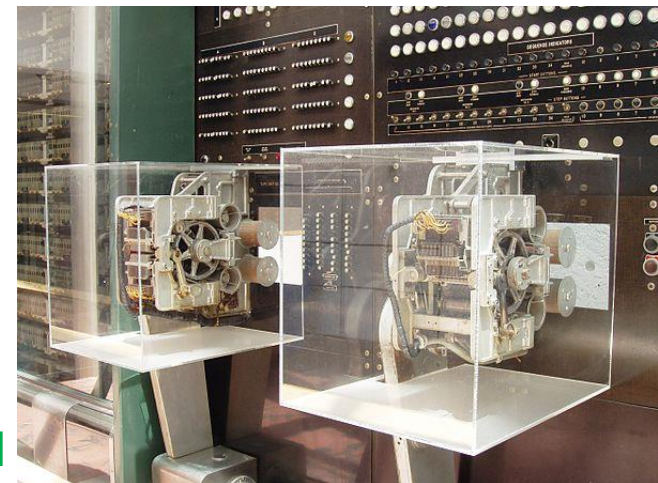
Πολλές ΓΠ ή θεωρητικά μοντέλα που δεν είχαν αναγκαστικά στόχο την υλοποίηση:

- **Plankalkül** (*Konrad Zuse*, υλοποίησε τον Η/Υ **Z3** το 1941...)

$$\begin{array}{l|l} & A + 1 \rightarrow A \\ v & 4 \quad 5 \\ s & 1.n \quad 1.n \end{array} \quad \longleftrightarrow \quad A(5) := A(4) + 1$$



- **Turing Machine** (*Alan Turing*)
- **$\lambda$ -calculus** (*Alonzo Church*)
- **Διαγράμματα Ροής** (*Von Neumann*)
- **IBM Mark I** (1944, Harvard, *Howard Aiken*)



IBM Mark I

# Ιστορικά (2)

## ■ Μέσα δεκαετίας του 1950

### □ FORTRAN (FORmula TRANslation)

IBM (1954) για τον IBM 704, John Backus (Υλοποίηση 1957)

- Χωρίς συναρτήσεις του χρήστη
- Μεταβλητές 2 char
- If <συνθήκη> 8, 10 (δηλ. If <συνθήκη> true goto 8, else goto 10)

## ■ Αρχές δεκαετίας του 1960

### □ ALGOL 60 (Algorithmic Language 1960)

Backus, Bauer, Green, Katz, etc. (ΗΠΑ – Ευρώπη, 1960)

- Αλγοριθμική γλώσσα περιγραφής υπολογιστικών διεργασιών
- Επιτακτική (imperative): Αλγόριθμος ως ακολουθία αλλαγών μνήμης
- Βασικές μονάδες υπολογισμού: blocks, procedures
- Έννοιες «Τύπου» και «Ελέγχου Τύπων»
- Κανόνες λεξικής εμβέλειας

# Ιστορικά (3)

- **COBOL** (COmmon Business Oriented Language)  
1960, Υπ. Άμυνας ΗΠΑ
  - Περιγραφή δεδομένων ανεξάρτητη από μηχανή (Συστήματα Διαχείρισης Βάσεων Δεδομένων)
  - Γενική if then else
  - Σχόλια
- Δεκαετία 1960 και στη συνέχεια... (βάση η ALGOL)
  - **BASIC** (Beginners All purpose Symbolic Instruction Code)  
1964, Dartmouth College, USA (Interpreter το 1975 η Microsoft)
  - **Pascal**  
1971, Niklaus Wirth
  - **C**  
1972, Bell Labs, Dennis Ritchie (για UNIX system programming)

# Ιστορικά (4)

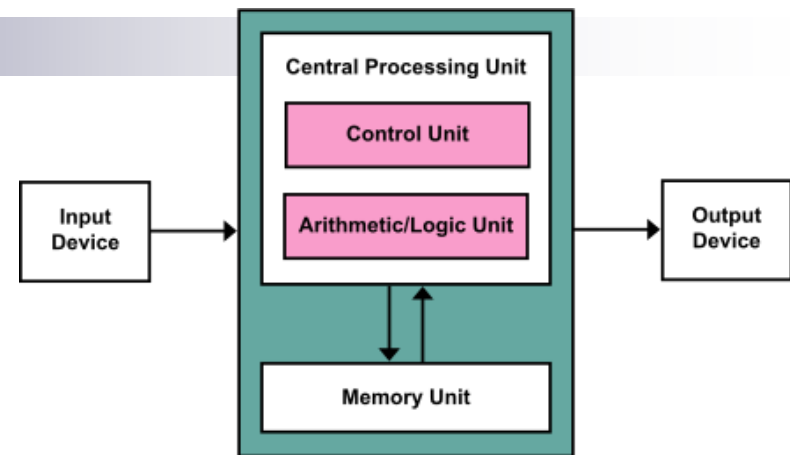
- C++  
1979, Bell Labs, Bjarne Stroustrup
- Java  
1991, Sun
- Python  
1991, Guido van Rossum
- HTML (HyperText Markup Language) – όχι ακριβώς ΓΠ...  
1991, CERN, Tim Berners-Lee
- PHP (Hypertext Preprocessor)  
1994, Rasmus Lerdorf
- JavaScript  
1995, Netscape
- .....

Εντωμεταξύ:

Ada, APL, LISP, SNOBOL, PL/1, SIMULA, ALGOL-W, Scheme, ML, Prolog, Smalltalk, VisiCalc, RPG, Perl, ...



# Ομάδες Γλωσσών



Von Neumann architecture

## ■ Επιτακτικές (Imperative) Γλώσσες Προγραμματισμού

“Pure” Von Neumann	FORTRAN, Pascal, BASIC, C
Αντικειμενοστραφείς	C++, Java, Smalltalk, Python
Γλώσσες Περιγραφής Σεναρίων (Scripting Languages)	JavaScript / NodeJS, PHP, Perl

## ■ Γλώσσες Εφαρμοστικού (Applicative) Προγραμματισμού

Συναρτησιακές (Functional)	Scheme, ML, LISP, Python, Scala
Λογικές (Logic)	Prolog, VisiCalc, RPG

# Επιτακτικές Γλώσσες (1)

- Ο υπολογισμός επιτυγχάνεται με αλλαγή κατάστασης μεταβλητών (θέσεις μνήμης).
- Οι πιο δημοφιλείς ΓΠ.
- Δυνατότητες:
  - Ενοποίηση ομάδας εντολών (begin-end, { })
  - Περιγραφή ομάδων δεδομένων (arrays, records)
  - Περιγραφή ροής ελέγχου (while, for, case)
  - Μηχανισμοί αφαίρεσης (abstraction) δεδομένων (function, procedure)
- Δεν εκμεταλλεύονται πλήρως τις δυνατότητες των Η/Υ.  
Επεξεργασία μιας λέξης τη φορά

# Επιτακτικές Γλώσσες (2)

## ■ “Von Neumann bottleneck”

Π.χ. για την πρόσθεση 2 μεταβλητών:

1. Πρόσβαση στη διεύθυνση 1<sup>ης</sup> μεταβλητής
2. Λήψη της τιμής που υπάρχει στη διεύθυνση 1<sup>ης</sup> μεταβλητής
3. Πρόσβαση στη διεύθυνση 2<sup>ης</sup> μεταβλητής
4. Λήψη της τιμής που υπάρχει στη διεύθυνση 2<sup>ης</sup> μεταβλητής
5. Πρόσθεση των δύο αριθμών
6. Πρόσβαση στη διεύθυνση 3<sup>ης</sup> μεταβλητής
7. Αποθήκευση αποτελέσματος πρόσθεσης στη διεύθυνση 3<sup>ης</sup> μεταβλητής

Δηλαδή, χρειάζονται 6 προσπελάσεις στη μνήμη, ενώ η CPU πρέπει να περιμένει την περισσότερη ώρα ...

# Γλώσσες Συναρτησιακού Προγραμματισμού (Functional Programming Languages)

- Βασίζονται σε μαθηματικές συναρτήσεις
- Δεδομένα και προγράμματα έχουν την ίδια μορφή
- Δεν χρησιμοποιούνται «μεταβλητές» και εντολές ανάθεσης
- Επανάληψη με χρήση αναδρομής

## LISP:

(print(process\_data(get\_data(...))))



begin
GetData(...);
ProcessData(...);
OutputData(...);
end.

# Γλώσσες Λογικού Προγραμματισμού (Logic Programming Languages)

- Προγράμματα σε μορφή συμβολικής λογικής
- Δηλώνονται οι προδιαγραφές των επιθυμητών αποτελεσμάτων
- Ουσιαστικά δημιουργούνται Βάσεις Δεδομένων
- Προγράμματα: facts, rules, queries

## Prolog:

**FACT:** mother (Helen, Jim)

**RULE:** grandparent (X, Z) : -parent (X, Y), parent (Y, Z)

**QUERY:** father (Bob, Joe)

# Κριτήρια καλού σχεδιασμού μιας ΓΠ

- Καλή συντακτική και σημασιολογική περιγραφή
- Αξιοπιστία
- Γρήγορη μετάφραση
- Αποδοτικός αντικειμενικός κώδικας
- Ορθογωνιότητα
- Ανεξαρτησία από μηχανή
- Τεκμηρίωση
- Συνέπεια

# Για την **επιτυχία** μιας ΓΠ

- Απλότητα
- Σαφήνεια συντακτικού
- Υποστήριξη αφαιρετικότητας
- Ευκολία πιστοποίησης ορθότητας προγραμμάτων
- Περιβάλλον ανάπτυξης εφαρμογών
- Κόστος χρήσης
  - εκτέλεση
  - μετάφραση
  - δημιουργία
  - συντήρηση