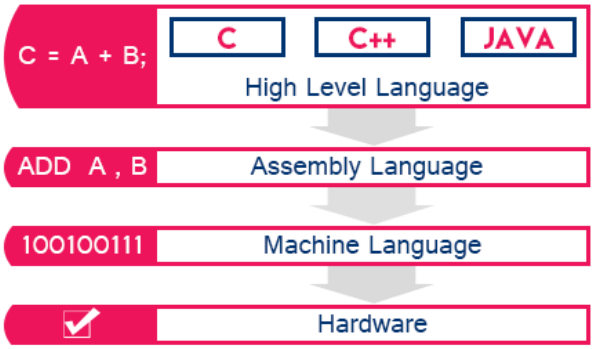
Despre limbaje de programare

Limbajele de programare se pot clasifica după diverse criterii.

Limbaj low level / high level (de nivel înalt)

Limbajele se pot clasifica în funcție de **apropierea de limbajul mașină** (după **gradul de abstractizare**) astfel:

- **Limbaj mașină** – cod binar, comunică direct cu hardware
- **Limbaj de asamblare** – o îmbunătățire: abrevieri de cuvinte în engleză pentru a specifica instrucțiuni (nu cod binar) => este necesar un "traducător" în cod mașină (Assambler)
- **Limbaj high level** – apropiat limbajului natural (uman); folosesc verbe pentru a desemna acțiuni (**do, repeat, read, write, continue, switch, call, goto**, etc.), conjuncții (**if, while**), adverbe (**then, else**)...



https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-2-9-class_prog_langs.html

LIMBAJ LOW LEVEL	LIMBAJ HIGH LEVEL
rapide, utilizare eficientă a memoriei, comunică direct cu hardware	mai ușor de utilizat, de detectat erori, nivel mai mare de abstractizare
nu necesită compilator/interpretor	mai lente, trebuie traduse în cod mașină
cod dependent de mașină, greu de urmărit	independente de mașină, pot fi portabile
programatorul are nevoie de cunoștințe legate de arhitectura calculatorului pe care dezvoltă programul	nu comunică direct cu hardware => folosesc mai puțin eficient resursele
utilizate pentru dezvoltare SO, aplicații specifice	arie largă de aplicații

Limbaj compilat / limbaj interpretat

Intuitiv – Pentru a citi o carte scrisă într-o limbă pe care nu o știm fie avem varianta cărții **tradusă deja** în limba noastră (pentru noi, de "compilator", care poate fi citită doar de cei care știu această limbă), fie avem un traducător care ne **traduce cartea originală (**astfel, este distribuită cartea originală către orice cititor/calculator, iar traducătorul/interpretorul o traduce în limba pe care o cunoaștem noi/sistemul de operare => mai lent)

- **Un compilator** traduce codul sursă high-level **în cod mașină (low-level, binar) /executabil** , rezultând un fișier care poate fi rulat pe sistemul de operare **pe care a fost creat** (tradus)

- **Un interpretor** traduce și execută instrucțiune cu instrucțiune

Nu se poate face mereu o încadrare strictă.
Chiar dacă limbajul este interpretat, poate exista o fază de compilare care are ca rezultat un fișier intermediar/bytecode **portabil**, cu instrucțiuni pentru mașina virtuală.

LIMBAJ COMPILAT	**LIMBAJ INTERPRETAT**
compilatorul "traduce" tot programul, dar după execuția este mai rapidă (poate fi executat de mai multe ori dacă nu se modifică sursa)	este mai lent
erorile sunt semnalate la finalul compilării (în procesul de compilare nu sunt executate instrucțiunile)	procesul de interpretare (cu executare instrucțiuni) se oprește la prima eroare
se distribuie executabilul (rezultatul compilării), nu sursa	nu este generat executabil, se distribuie sursa
nu este portabil (executabilul se poate rula doar pe aceeași platformă)	este portabil este suficient să avem interpretorul (el e dependent de platformă)
trebuie recompilat după modificări	nesiguranța suportului
Exemple: C, C++	**Exemple: JavaScript, PHP, Java?**

Paradigme de programare

Limbajele se pot clasifica și în funcție de stilul de programare și facilitățile oferite (controlul fluxului, modularitate, clase etc) = paradigme de programare

- **Programare imperativă**
  - cea mai veche
  - programatorul dă instrucțiuni mașinii (care trebuie executate în ordine) despre pașii care trebuie să îi execute
  - cod mașină, Fortran, C, C++ etc
  - **Programare procedurală**
    - program modularizat, bazat pe apeluri de proceduri
    - Exemple: C, Pascal
  - **Programare orientată pe obiecte**
    - bazată pe conceptul de obiecte (care interacționează)
    - Exemple: C++, Java, C#
  - **Programare paralelă**
    - programul poate fi împărțit în seturi de instrucțiuni ce pot fi executate în paralel pe mai multe mașini
    - Exemple: Go, Java, Scala

- **Programare declarativă**
  - programatorul declară proprietăți ale rezultatului dorit, nu cum se obține rezultatul
  - **Programare logică**
    - Rezultatul este răspunsul la o interogare a unui sistem de date și reguli (bază de cunoștințe); execuția înseamnă activarea unui proces deductiv (bazat pe logică).
    - Exemple: Prolog
  - **Programare funcțională**
    - rezultatul este definit ca valoare a aplicării succesive ale unor funcții (matematice)
    - Exemple: Haskell, Lisp, Scala
  - **Programare la nivelul bazelor de date**
    - programul rezolvă cerințele unei gestiuni corecte și consistente a bazelor de date.
    - Ex.: FoxPro, SQL

Limbajul Python

Câteva caracteristici:

- high – level
- interpretat...
- hibrid – mai multe paradigme de programare:
  - procedural: subprograme și module
  - orientat obiect: clase
  - funcțional: funcții ca argumente ale altor funcții (filter, map, lambda)
- tip dinamic: variabilele nu au tip de date static (valorile au tip), li se pot asocia valori de tipuri diferite pe parcursul execuției programului: astfel, unei variabile i se poate schimba tipul pe parcursul execuției programului:

```
x = 10
x = "sir"
```

- orice valoare este un obiect, variabilele sunt referințe spre obiecte (nume pentru obiecte)
- Garbage collector

Avantaje

- sintaxa simplă, sugestivă
- dinamic
- de actualitate
- numeroase facilități (incluse automat): dezvoltare software, web, GUI, module pentru IA, ML (Google – motoare de căutare)
- portabil
- open-source: www.python.org
- garbage collection

Dezavantaje

- mai lent – high level, interpretat
- nu are atât de multe biblioteci ca alte limbaje, nu are suport pentru mobile
- nu verifică tipurile de date ale variabilelor la compilare
- nu folosește bine facilități precum procesoare multi-core

Scurt istoric

- conceput de Guido van Rossum (în C) 1980
- implementarea a început în decembrie 1989
- lansări majore:
  - Python 1.0: ian. 1994
  - Python 2.0: oct. 2000
  - **Python 3.0**: dec. 2008
  - Python 3.8: 2019
  - Python 3.x nu este 100% compatibil cu Python 2.x (!portabilitate, rescriere, nu mai este asigurat suport pentru Python 2 => dezavantaje open source)

```
print "Pyhon 2"
print("Pyhon 3")
```

  - **Vom folosi Python 3**.x
- este denumit după trupa de comedie / serialul BBC al anilor ’70 Monty Python
- Tim Peters (unul dintre creatorii limbajului) – a elaborat set de principii ale limbajului, îl putem afla cu instrucțiunea

```
import this
```

Instalare, rulare

- https://www.python.org/downloads/
- mai multe variante pot coexista
- variabila de mediu PATH – bifați la instalare opțiunea Add python to PATH
- linie de comanda: comanda python => interpretor

```
>>> import this
>>> print("Python 3")
```

- Medii de dezvoltare IDE : PyCharm, Spyder etc
- Fișier cu instrucțiuni – pe moodle