**Лекція 01 Історія розвитку мов програмування. Основні поняття програмування**

**Посилання на всі лекції та лабораторні роботи**

**https://github.com/TetyanaLumpova/OP-AM/tree/main/2022**

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

1. Основні визначення.
2. Історія розвитку мов програмування
3. Покоління мов програмування
4. Класифікація мов програмування.
5. Транслятор. Компілятор. Інтерпретатор.

**Мова програмування** — формальна знакова система, призначена для опису алгоритмів у формі, яка зручна для виконавця (наприклад, комп’ютера).

Мова програмування визначає набір лексичних, синтаксичних правил, які використовуються при складанні комп’ютерної програми. Вона дозволяє програмісту точно визначити те, на які події буде реагувати комп’ютер, як будуть зберігатись і передаватись дані, а також які саме дії слід виконати над цими даними при різних обставинах.

З часів створення перших комп’ютерів , людство винайшло вже більше ніж 2500 мов програмування. Деякими мовами вміє користуватися тільки невелика кількість їх розробників, інші стають відомими для мільйонів людей. Професійні програмісти іноді застосовують у своїй роботі більше десятка різноманітних мов програмування.

Алгоритмічні мови, призначені для побудови описів алгоритмів, що виконуються електронними обчислювальними машинами, називаються *мовами програмування*.

Описи алгоритмів мовою програмування називають *програмами*.

Вибір мови залежить від

1) специфіки задачі,

2) апаратної частини ПК та програмної платформи,

3) кваліфікації програміста.

Всі існуючи мови програмування можна поділити на дві групи:

• мови низького рівня;

• мови високого рівня.

До мов низького рівня належать мови асемблера (від англ. to assemble - складати, компонувати). У мові асемблера використовуються символьні позначення команд, які легко зрозуміти і запам'ятати. Замість послідовностей двійкових кодів команд записуються їх символьні позначення, а замість двійкових адрес даних, які використовуються під час виконання програми, символьні імена цих даних. Іноді мову асемблера називають мнемокодом або автокодом. Окрім того, ще виділяють мови низького рівня - машинні мови, які потребують вказівки дрібних деталей процесу обробки даних.

Більшість програмістів при складанні програм користуються мовами високого рівня. Мови ж високого рівня імітують природні мови, використовуючи деякі слова размовної мови і загальноприйняті математичні символи. Ці мови більш зручні для людей. Для опису використовується алфавіт мови. З цих символів складаються так звані службові слова мови, кожне з яких має певне призначення. Службові слова зв'язуються одне з одним в речення за певними синтаксичними правилами мови і визначають деяку послідовність дій, які мусить виконати комп'ютер.

Та програми, що написані на мовах програмування високого рівня (алгоритмічних мовах програмування), комп'ютер "не розуміє". Для того, щоб він міг виконати програму, її потрібно перекласти на машинну мову. Для такого перекладу використовують спеціальні програми – компілятори та інтерпретатори.

**Історія розвитку мов програмування**

Програми для перших обчислювальних машин створювались, як правило, в машинних кодах або на асемблері і були схожі на витвір мистецтва, бо повинні були поміститись у мініатюрному за сучасними поняттями об’ємі пам’яті.

**1930-1940 рр.** виникає лямбда-числення та машина Тюрінга, які застосували математичну абстракцію для опису алгоритмів. Лямбда-числення згодом здійснило вплив на проектування мов програмування.

Формальні визначення алгоритму з'явилися в тридцятих-сорокових роках 20 століття. Одним із перших було визначення англійського математика Алана Тюрінга, який у 1936 році описав схему деякої гіпотетичної (абстрактної) машини і запропонував називати алгоритмами те, що вміє робити така машина. При цьому визначенні, якщо щось не може бути зроблено машиною Тюрінга, це вже не алгоритм. Інакше кажучи, Тюрінг формалізував правила виконання дій за допомогою опису роботи деякої конструкції. **Маши́на Тю́рінга** — математичне поняття, введене для формального уточнення інтуїтивного поняття **алгоритму**. Аналогічну конструкцію машини згодом і незалежно від Тюрінга ввів американський математик Еміль Пост.[

**Ля́мбда-чи́слення**, або **λ-числення** — формальна система, що використовується в теоретичній кібернетиці для дослідження визначення функції, застосування функції, та рекурсії. Це числення було запропоноване [Алонсо Черчем](https://uk.wikipedia.org/wiki/%D0%A7%D0%B5%D1%80%D1%87_%D0%90%D0%BB%D0%BE%D0%BD%D1%81%D0%BE) та Стівеном Кліні в 1930-ті роки, як частина більшої спроби розробити базис математики на основі функцій, а не множин.

### ****1940-і рр.**** створюються перші електричні, двійкові комп'ютери. Вважається, що першу мову програмування високого рівня – Планкалькюль (нім. - Plankalkül) розробив Конрад Цузе в період 1943 – 1945 рр. але в той час вона не була реалізована.

### В кінці 40-х — початку 50-х р.р. застосовувалися інтерпретовані системи кодування, коли певні команди мови програмування кодувалися числами, які уже інтерпретувалися машинним кодом. Це отримало назву «автоматичного програмування» і було простішими для програмування, ніж машинні коди, але могло мати значно меншу (до 50 разів) швидкодію, через що все ж таки часто надавали перевагу машинним кодам.

**При кодуванні** безпосередньо в машинних кодах, а основним носієм інформації були перфокарти та перфострічки. Програмісти повинні були знати архітектуру машини досконально. Програми були досить простими, що було обумовлено, по-перше, всіма обмеженими можливостями цих машин, та, по-друге, великою складністю розробки та, головне, налагодження програм безпосередньо на машинних мовах. Разом з тим такий спосіб розробки давав програмісту просте величезну владу над системою..

Першим значним кроком став перехід до **мови ассемблера** (англійська мова assembly language, або assembler). Цей перехід до символьного кодування машинних команд – мав насправді величезне значення. Програмісту не треба було більше занурюватися до способів кодування команд на апаратному рівні. Більш того, зазвичай однакові за суттю команди кодувались зовсім різними способами залежно від власних параметрів. З’явилася також можливість використання макросів та міток, що також спрощувало створення, модифікацію та налагодження програм. З’явилася навіть деяка сумісність цілого сімейства машин зі схожою системою команд та деякого загального ассемблера для них, при цьому не було потреби забезпечувати двійкову сумісність. Вперше в історії розвитку програмування з’явилося два представлення програми: у вихідних текстах та у відкомпільованому вигляді. До кінця ассемблерної ери можливість автоматичної трансляції в обидва боки була втрачена назавжди.

### ****1957 рік**** Першою широковживаною компільованою мовою став розроблений групою Джона Бекуса Fortran, анонсований у 1954 році та випущений у 1957 для ІВМ 704. Основним призначенням Фортрану були швидкі наукові обчислення, оголошувалося що швидкодія згенерованого компілятором коду майже не відрізянтиметься від машинного коду написаного вручну. Уже у квітні 1958 близько половини програм для IBM 704 були написані на Фортрані.

Це перший крок програмування високого рівня. Вперше програміст міг по справжньому абстрагуватися від особливостей машинної архітектури. Ключовою ідеєю, що відрізняв нову мову від ассемблера, була **концепція підпрограм**. Крім того, синтаксична структура мови була достатньо складна для машинної обробки в першу чергу через те, що пробіли як синтаксичні одиниці взагалі не використовувались. Це породжувало масу помилок. Мова Фортран використовувалася (і використовується на сьогоднішній день) для наукових обчислень. Відсутні звичні мовні конструкції, семантична перевірка компілятором, підтримка сучасних способів структурування коду та даних. Це усвідомлювали і самі розробники. За визнанням самого Бекуса, перед ними стояла задача скоріше розробки компілятора, аніж мови. Розуміння самостійного значення мов програмування прийшло пізніше. Появу Фортрана зустріли ще з більшою критикою, аніж ассемблера. Програмістів лякало зниження ефективності програм за рахунок використання проміжної ланки у виді компілятора. І ці побоювання мали під собою основу: дійсно, гарний програміст, скоріш за все, при розв’язуванні якої-небудь невеликої задачі вручну напише код, що працюватиме швидше, аніж код, отриманий як результат компіляції. Через деякий час прийшло розуміння того, що реалізація великих проектів неможлива без використання мов високого рівня. Потужність обчислювальних машин зростала, і переваги мов високого рівня стали настільки очевидними, що підштовхнуло розробників до творення нових мов, все більш сучасних.

### ****1958 рік**** у MIT (Массачусетський технологічний інститут) розробили ****LISP**** (LISP, від англ. LISt Processing — “обробка списків”; сучасне написання: Lisp)– першу функційну мову, яка понад 25 років домінувала у програмуванні задач штучного інтелекту. Lisp — сімейство мов програмування, програми і дані в яких представляються системами лінійних списків.

**1960 рік** - створена мова програмування **Cobol**. Вона задумувалася як мова для створення комерційних додатків, і він став таким. На Коболі написані тисячі прикладних комерційних систем. Відмітною особливістю мови є можливість ефективної роботи з великими масивами даних, що характерно саме для комерційних прикладних задач. Популярність Кобола настільки висока, що навіть зараз, при всіх його недоліках (за структурою та задуму Кобол нагадує Фортран) з’являються нові його діалекти та реалізації. Так нещодавно з’явилась реалізація Кобола, у поєднанні з Microsoft.NET, що вимагало внесення до мови деяких рис об’єктно-орієнтованої мови.

**1960 рік** - командою на чолі з Петером Науром (Peter Naur) була створена мова програмування Algol. Ця мова дала початок цілому сімейству Алгол-подібних мов (найважливіший представник - Pascal). У 1968 році з'явилася нова версія мови. Вона не знайшла такого широкого практичного застосування, як перша версія, але була вельми популярна в кругах теоретиків. Мова була достатньо цікава, оскільки володіла багатьма унікальним на той момент характеристиками.

### ****1963 рік**** Дартмурським коледжом (Dartmouth College) створено спрощену мову BASIC (Beginners All-purpose Symbolic Instruction Code – багатоцільова мова символічних інструкцій для початківців) для навчання програмуванню студентів, які переважно спеціалізувалися у вільних мистецтвах, а не технічних науках. Мова замислювалася в першу чергу як засіб навчання і як перша мова програмування, що вивчалася. Він передбачався таким, що легко інтерпретується і компілюється. Було створено декілька потужних реалізацій BASIC, що підтримують найсучасніші концепції програмування (наприклад, Microsoft Visual Basic).

### ****1964 рік**** корпорація IBM створила мову PL/1, яка була покликана замінити Кобол та Фортран. Мова мала виняткове розмаїття синтаксичних конструкцій. В ній вперше з’явилася обробка виняткових ситуацій та підтримка паралелізму. Треба відмітити, що синтаксична структура мови була украй складною. Пропуски вже використовувалися як синтаксичні роздільники, але ключові слова не були зарезервовані.

Створення кожна з вищезазначених мов (за виключенням, можливо, Algol’а) було викликано деякими практичними вимогами. Ці мови послужили фундаментом для пізніших розробок. Всі вони представляють одну і ту ж парадигму програмування. Наступні мови пішли істотно далі в своєму розвитку, убік глибшого абстрагування.

**Кінець 60-х років** - **Си́мула-67** (Simula 67) — перша об’єктно-орієнтовна мова програмування. Ця мова у значній мірі випередила свій час, сучасники (програмісти 60-х років) виявились не готові сприйняти цінність мови Simula 67, і він не витримав конкуренції з іншими мовами програмування (пред усім, з мовою Fortran).

**1971 рік** Ніклаус Вірт опублікував опис мови [Pascal](https://youtu.be/MS43T6CI2zI), яка у 70-их стала загальновживаною для навчання студентів. Назва мові дана на честь видатного французького математика, фізика, літератора і філософа Блеза Паскаля. Мова чудова тим, що це перша широко поширена мова для структурного програмування (першим, строго кажучи, був Алгол, але він не набув такого широкого поширення). Вперше оператор безумовного переходу перестав грати основоположну роль при управлінні порядком виконання операторів. У цій мові також упроваджена строга перевірка типів, що дозволило виявляти багато помилок на етапі компіляції. Негативною рисою мови була відсутність в ній засобів для розбиття програми на модулі. Вірт усвідомлював це і розробив мову ***Modula-2*** (1978), в якій ідея модуля стала однією з ключових концепцій мови. У 1988 році з'явилася ***Modula-3***, в яку були додані об'єктно-орієнтовані риси. Логічним продовженням Pascal і Modula є мова ***Oberon*** і ***Oberon-2***. Вони характеризуються рухом убік об’єктно- і компоненто- орієнтованості.

### ****1972 рік**** Деніс Річі розробив у Bell Labs мову ****C****. Тоді ж створено інтерпретатор мови Пролог (Prolog) – першої і найвідомішої мови логічного програмування. Пролог сприймає у якості програми деякі описи задач, і сам робить пошук вирішення.Алан Кей у Xerox PARC розробив першу широко вживану об'єктно-орієнтовану мову – Smalltalk.

Мова C створювалася як мова для розробки операційної системи UNIX. *C* часто називають «асемблером, що переноситься», маючи на увазі те, що він дозволяє працювати з даними практично так само ефективно, як на асемблері, надаючи при цьому структуровані конструкції, що управляють, і абстракції високого рівня (структури і масиви). Саме з цим пов'язана його величезна популярність і понині. І саме це є його ахіллесовою п'ятою. Компілятор *C* дуже слабо контролює типи, тому дуже легко написати зовні абсолютно правильну, але логічно помилкову програму.

### ****1975 рік**** в Массачусетському технологічному інституті описано спрощений діалект мови Лісп – [Scheme](https://youtu.be/AqBxU-Zmx00).

### 1983 рік під егідою Міністерства Оборони США була створена мова Ada у результаті проекта, прийнятого Міністерством оборони США з ціллю розробити єдину мову програмування для бортових систем управління воєнними об’єктами (кораблями, літаками, танками, ракетами, снарядами і т. п.).Мова чудова тим, що дуже багато помилок може бути виявлено на етапі компіляції. Крім того, підтримуються багато аспектів програмування, які часто віддаються на відкуп операційній системі (паралелізм, обробка виключень). У 1995 році був прийнятий стандарт мови Ada 95, яка розвиває попередню версію, додаючи в неї об'єктну орієнтованість і виправляючи деякі неточності. Обидві ці мови не отримали широкого розповсюдження поза військовими й іншими великомасштабними проектами (авіація, залізничні перевезення). Основною причиною є складність освоєння мови і достатньо громіздкий синтаксис (значно громіздкіший, ніж Pascal).

### ****1984 рік**** з метою об'єднання різних діалектів Ліспу створено Common Lisp. Випущено MATLAB.

### ****1985 рік**** Б'ярн Страуструп опублікував реалізацію мови C++, додавши в мову C об'єктно-орієнтовані риси, узяті з Simula, і виправивши деякі помилки і невдалі рішення мови. C++ продовжує удосконалюватися, 4 вересня 2020 року затверджено Стандарт C++20.

### ****1987 рік**** створено ****Perl****. Автор - Ларрі Уолт. Саме слово Perl — аббревіатура, котра розшифровується як Practical Extraction and Report Language (практична мова звітів, через що спочатку називалась PEARL, але потім буква “A” “загубилась”). Талісманом мови Perl є верблюд — не надто гарне, проте дуже витривале створіння, здатне виконувати тяжку роботу. Основною особливістю цієї мови вважають його багаті можливості для роботи з текстом. Perl — платформа стійкої, перспективної мови програмування. Він використовується для маси проектів у публічних та приватних секторах і широко використовується для додатків мережних програм усіх потреб.

### ****1990 рік**** з’являється мова функціонального програмування **Haskell**.

### ****1991 рік**** створено [Visual Basic](https://docs.microsoft.com/ru-ru/visualstudio/python/tutorial-working-with-python-in-visual-studio-step-01-create-project?view=vs-2019) і опубліковано [Python](https://pythonworld.ru/samouchitel-python).

### ****1992 рік**** випущено Oracle 7 з підтримкою PL/SQL.

### ****1995 рік**** Sun Microsystems випустила Java, Netscape – JavaScript, тоді ж створено PHP і Ruby.

Мова ***Java*** створена Кеном Арнольдом і Джеймсом Гослінгом. Вона успадковувала синтаксис C і C++ і була позбавлена від деяких неприємних рис останнього. Відмітною особливістю мови є компіляція в код якоїсь абстрактної машини, для якої потім пишеться емулятор (Java Virtual Machine) для реальних систем. Крім того, в Java немає показників і множинного спадкоємства, що сильно підвищує надійність програмування.

З самого початку нова мова програмування називалась JaGo (James Gosling) і розроблялась для побутової електроніки, але згодом була перейменована на Java і стала використовуватись для написання додатків і серверного програмного забезпечення.

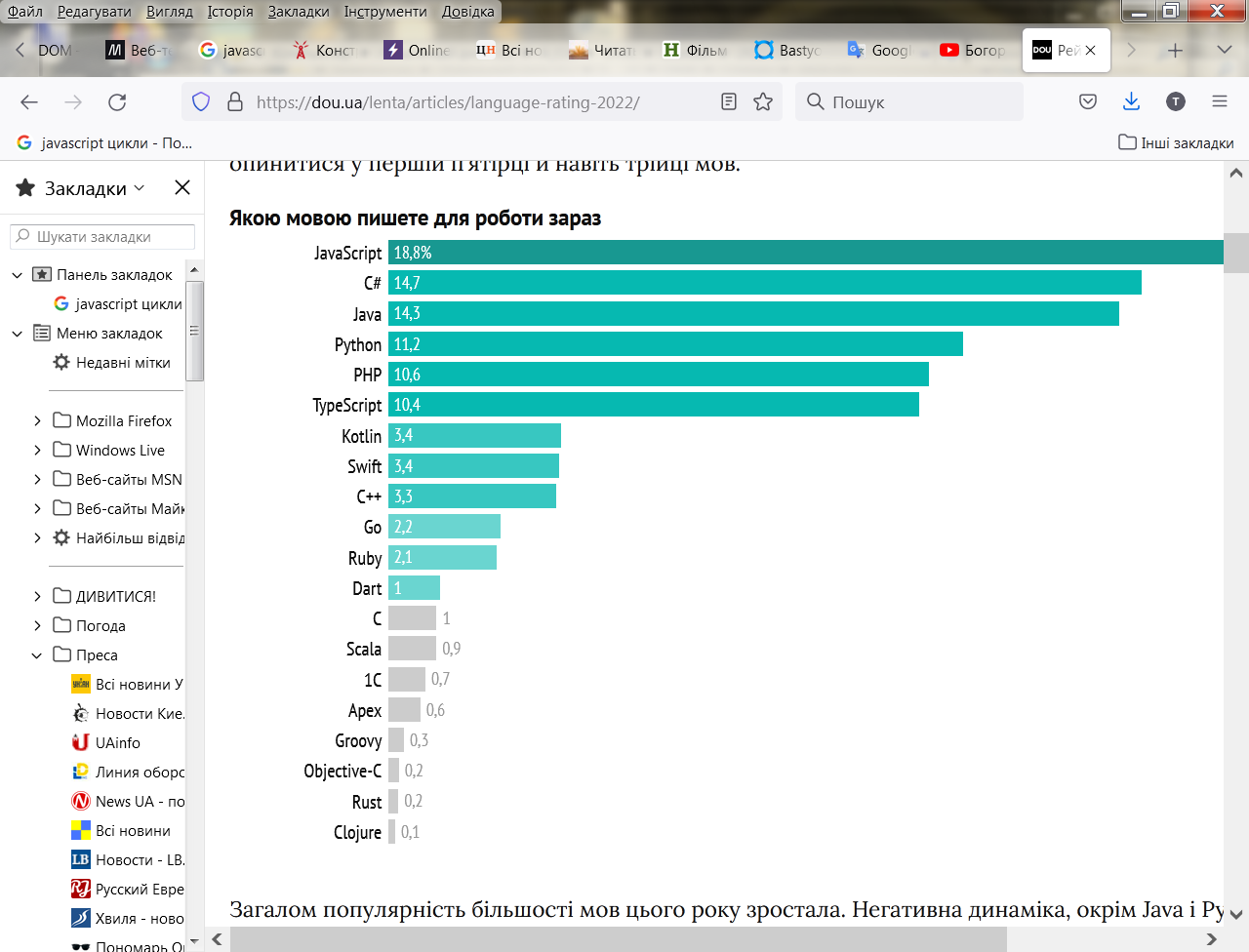
### 1999-2000 ****роки**** в корпорації Microsoft була створена мова *C#*. Вона достатньою мірою схожа з Java (і замислювалася як альтернатива останній), але має і відмітні особливості. Орієнтована, в основному, на розробку багатокомпонентних Інтернет-додатків.

### ****2007 рік**** компанією Google розпочато розробку мови ****Go****.

### ****2010 рік**** компанією JetBrains розпочато розробку мови [*Kotlin*](https://kotlinlang.ru/)

### ****2014 рік**** компанією Apple преставлено мову Swift.

Найбільш перспективними і затребуваними мовами програмування на 2020 рік є Python, Java, С, Javascript і С ++

****

Дані на 2022 різ з сайту https://dou.ua/lenta/articles/language-rating-2022/

**Довідково з Вікіпедії.**

**Джон фон Не́йман** чи **Джон фон Но́йман** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *John von Neumann*), **Нейман Янош Лайош** ([угор.](https://uk.wikipedia.org/wiki/%D0%A3%D0%B3%D0%BE%D1%80%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Neumann János Lajos*; **Йоганн фон Нойман** ([нім.](https://uk.wikipedia.org/wiki/%D0%9D%D1%96%D0%BC%D0%B5%D1%86%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Johann von Neumann*; [28 грудня](https://uk.wikipedia.org/wiki/28_%D0%B3%D1%80%D1%83%D0%B4%D0%BD%D1%8F) [1903](https://uk.wikipedia.org/wiki/1903) — [8 лютого](https://uk.wikipedia.org/wiki/8_%D0%BB%D1%8E%D1%82%D0%BE%D0%B3%D0%BE) [1957](https://uk.wikipedia.org/wiki/1957)) — американський [математик](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA) [угорського](https://uk.wikipedia.org/wiki/%D0%A3%D0%B3%D0%BE%D1%80%D1%86%D1%96) походження, що зробив значний вклад у [квантову фізику](https://uk.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0_%D1%84%D1%96%D0%B7%D0%B8%D0%BA%D0%B0), [функціональний аналіз](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7), [теорію множин](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD), [інформатику](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), [економічні](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D1%96%D0%BA%D0%B0) науки та в інші численні розділи знання. Він став засновником [теорії ігор](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D1%96%D0%B3%D0%BE%D1%80) разом із [Оскаром Морґенштерном](https://uk.wikipedia.org/wiki/%D0%9E%D1%81%D0%BA%D0%B0%D1%80_%D0%9C%D0%BE%D1%80%D2%91%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D1%80%D0%BD) у [1944](https://uk.wikipedia.org/wiki/1944) році. **Розробив архітектуру (так звану** [**«архітектуру фон Неймана»**](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0)**), яка використовується в усіх сучасних** [**комп'ютерах**](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80)**.**

**Покоління мов програмування**

Розрізняють п'ять поколінь мов програмування

* ***Перше покоління.***

 Початок 1950-х років — мови перших комп’ютерів.

Перша асемблероподібна мова створена за принципом «одна інструкція — один рядок».  Основна відмінна риса: орієнтування на конкретний комп’ютер.

* ***Друге покоління***

Кінець 1950-х — початок 1960-х р.р.

Розроблено символьний асемблер, в якому з’явилося поняття змінної. Це перша повноцінна мова програмування. Основна відмінна риса: орієнтування на абстрактний комп’ютер з такою ж системою команд.

* **Третє покоління**

1960-ті р.р. — мови програмування високого рівня.

Їхні характеристики:

* відносна простота;
* незалежність від конкретного комп’ютера;
* можливість використання потужних синтаксичних конструкцій.

Простота мов дає змогу писати невеликі програми і людям, які не є професійними програмістами. Всього у світі існує близько 200 популярних мов програмування третього рівня.

* ***Четверте покоління***

Початок 1970-х р.р. до сьогоднішнього часу.

Створюються мови, призначені для реалізації великих проектів. Зокрема, це проблемно-орієнтовані мови, що оперують конкретними поняттями вузької галузі. Як правило, в такі мови вбудовують потужні оператори, що дозволяють одним рядком описувати функції, для опису яких мовами молодших поколінь потрібно було б сотні чи навіть тисячі рядків початкового коду. Приклади: SQL, HTML, XML, Prolog, та багато інших вузькоспеціалізованих декларативних мов.

Основна відмінна риса мови четвертого покоління: наближення до мови людини (декларативні мови).

Деякі мови мають риси одночасно і третього і четвертого поколінь.

* ***П'яте покоління***

П’ятого покоління мов програмування поки що не існує.

Виробники пропрієтарних програмних продуктів часто намагаються приписати своїм продуктам якісь маркетингові особливості, і деколи вказують що їхній продукт — це "мова п’ятого покоління". Насправді, всі ці продукти — це просто середовища для прискореного створення продуктів (Rapid Application Development - RAD), і використовують мови третього та четвертого поколінь.

На тенденції розробки мов програмування впливає:

* архітектура комп’ютерів (майже всі комп’ютери I – IV поколінь відповідають принципам організації ЕОМ Джона фон Неймана, який 1945 року запропонував наступні положення: 1) вся інформація представлена у вигляді сигналів, що відповідають двом станам 0 та 1 (є сигнал – немає сигналу, намагнічено – не намагнічено); 2) задана програма; 3) програма та дані зберігаються в комірках пам’яті; немає різниці між даними та програмою);
* методологія програмування
* 50-і – початок 60-х: прості задачі, намагання максимальної машинної віддачі;
* кінець 60-х – програмістська ефективність, читабельність, структурність;
* кінець 70-х – абстракція даних;
* середина 80-х – ООП.

**Класифікація мов програмування.**

**Мови обробки даних**. Майже всі перелічені вище мови є мовами загального призначення в тому сенсі, що вони не орієнтовані і не оптимізовані під використання яких-небудь специфічних структур даних або на застосування в яких-небудь специфічних областях. Було розроблено велику кількість мов, орієнтованих на достатньо специфічні застосування. У 1957 році була зроблена спроба створення мови для опису математичної обробки даних. Мова була названа ***APL (Application Programming Language)***. Її відмітною особливістю було використання математичних символів (що ускладнювало застосування на текстових терміналах; поява графічних інтерфейсів зняла цю проблему) і дуже могутній синтаксис, який дозволяв проводити безліч нетривіальних операцій прямо над складними об'єктами, не вдаючись до розбиття їх на компоненти. Широкому застосуванню перешкодило, як вже наголошувалося, використання нестандартних символів як елементів синтаксису. У 1962 році з'явилася мова ***Snobol*** (а в 1974 - його наступник ***Icon***), призначена для обробки рядків. Синтаксис Icon нагадує C і Pascal одночасно. Відмінність полягає в наявності потужних вбудованих функцій роботи з рядками і пов'язана з цими функціями особлива семантика. Сучасним аналогом Icon і Snobol є ***Perl*** - мова обробки рядків і текстів, в яку додано деякі об'єктно-орієнтовані можливості. Вважається дуже практичною мовою, проте їй бракує елегантність. У 1969 році була створена мова ***SETL*** - мова для опису операцій над множинами. Основною структурою даних в мові є множина, а операції аналогічні математичним операціям над множинами. Корисна при написанні програм, що мають справу з складними абстрактними об'єктами.

**Скриптові мови.** Останнім часом у зв'язку з розвитком Інтернет-технологій та продуктивності комп'ютерів набули поширення так звані скриптові мови. Ці мови спочатку орієнтувалися на використання в якості внутрішніх керуючих мов у складних системах. Багато хто з них, проте ж, вийшов за межі сфери свого початкового застосування і використовуються нині в зовсім інших областях. Характерними особливостями даних мов є, по-перше, їх інтерпретованість (компіляція або неможлива, або небажана), по-друге, простий синтаксис, а по-третє, легка розширюваність. Таким чином, вони ідеально підходять для використання в часто змінних програмах, дуже невеликих програмах або у випадках, коли для виконання операторів мови витрачається час, неспівставний з часом їх розбору. Було створено чималу кількість таких мов. Мова ***JavaScript*** була створена в компанії Netscape Communications як мова для опису складної поведінки веб-сторінок. Спочатку називалася LiveScript, причиною зміни назви стали маркетингові міркування. Інтерпретується браузером під час відображення веб-сторінки. За синтаксисом схожа з Java і (віддалено) з C/C++. Має можливість використовувати вбудовану в браузер об'єктну функціональність, проте об'єктно-орієнтованою мовою не є. Мова ***VBScript*** була створена в корпорації Microsoft багато в чому як альтернатива JavaScript. Має схожу область застосування. Синтаксично схожий з мовою Visual Basic (і є урізаною версією останньої). Так само, як і JavaScript, виконується браузером при відображенні веб-сторінок і має той ступінь об’єктно-орієнтованості. Мова ***Perl*** створювалася в допомогу системному адміністраторові операційної системи Unix для обробки різного роду текстів і виділення потрібної інформації. Розвинулася до потужного засобу роботи з текстом. Є мовою, що інтерпретується, і реалізована практично на всіх існуючих платформах. Застосовується при обробці текстів, а також для динамічної генерації веб-сторінок на веб-серверах. Об'єктно-орієнтована мова програмування ***Python***, що інтерпретується. За структурою та областю застосування близька до Perl, але строгіша й логічніша. Є реалізації для більшості існуючих платформ.

**Об’єктно-орієнтовані мови.** Об'єктно-орієнтований підхід, що прийшов на зміну структурному, вперше з'явився зовсім не в C++, як вважають деякі. Існує ціла низка чистих об'єктно-орієнтованих мов, без відомостей про які наш огляд був би неповним. Першою об’єктно-орієнтованою мовою була мова ***Simula*** (1967). Ця мова була призначена для моделювання різних об'єктів і процесів, і об'єктно-орієнтовані риси з'явилися в ній саме для опису властивостей модельних об'єктів. Популярність об'єктно-орієнтованому програмуванню принесла мова ***Smalltalk***, створену в 1972 році. Мова призначалася для проектування складних графічних інтерфейсів і була першою по-справжньому об'єктно-орієнтованою мовою. У ній класи і об'єкти - це єдині конструкції програмування. Великим недоліком Smalltalk є великі вимоги до пам'яті і низька продуктивність отриманих програм. Це пов'язано з не дуже вдалою реалізацією об'єктно-орієнтованих особливостей. Існує мова з дуже вдалою реалізацією об'єктної-орієнтованості, що не є надбудовою ні над якою іншою мовою – ***Eiffel*** (1986). Будучи чистою мовою об'єктно-орієнтованого програмування, вона, крім того, підвищує надійність програми шляхом використання „контрольних тверджень”.

Створення **UML** (Unified Modeling Language - Уніфікована мова моделювання) почалось у 1994 р. В цей час Грейді Буч (Grady Booch) та Джеймс Рембо (James Rambaugh) почали об’єднувати декілька методів об’єктно - орієнтованого моделювання в фірмі Rational Software. Та вже в 1995 р. було представлено специфікацію методу, названого Unified Method. Перша версія UML була прийнята консорціумом OMG (Object Management Group) в січні 1997 р. Затверджена в вересні версія UML 1.1 була прийнята основними компаніями – виробниками програмного забезпечення, такими, як Microsoft, IBM, Hewlett-Packard та виробниками CASE-засобів, які реалізовували підтримку UML в своїх програмних продуктах (Paradigm Plus, Microsoft Visual Modeler for Visual Basic, Delphi и др.)

**Мови паралельного програмування.** Більшість комп'ютерної архітектури і мов програмування орієнтовані на послідовне виконання операторів програми. В даний час, проте, існують програмно-апаратні комплекси, що дозволяють організувати паралельне виконання різних частин одного і того ж обчислювального процесу. Для програмування таких систем необхідна спеціальна підтримка з боку засобів програмування, зокрема, мов програмування. Деякі мови загального призначення містять в собі елементи підтримки паралелізму, проте ж програмування суто паралельних систем потребує спеціальних прийомів. Однією з таких мов є Occam, Linda.

**Неімперативні мови.** Всі мови, про які йшлося раніше, мають одну загальну властивість: вони імперативні. Це означає, що програми на них, зрештою, є покроковим описом рішення тієї або іншої задачі. Можна спробувати описувати лише постановку проблеми, а вирішувати задачу доручити компілятору. Існує два основні підходи, що розвивають цю ідею: функціональне і логічне програмування.

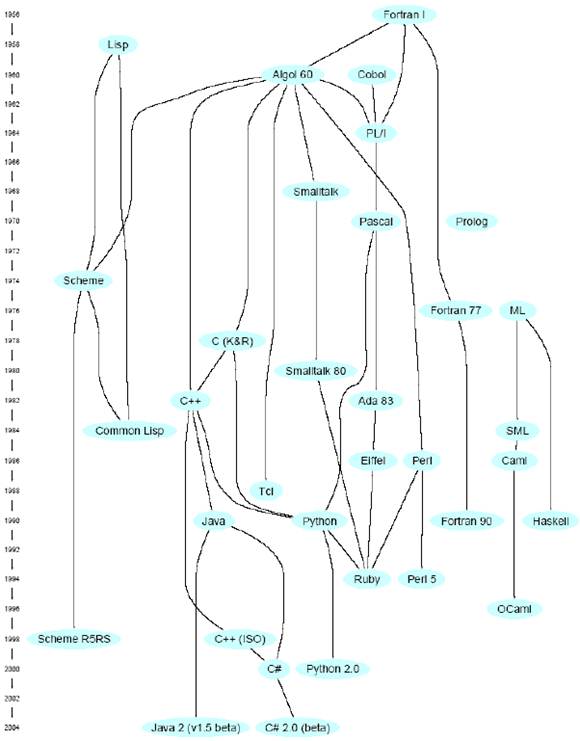
Таким чином;

1) імперативні (процедурні) – результат обчислення деякого виразу розміщується в області пам’яті, що представлена в програмі змінною (причому проміжні результати обчислення теж зберігаються в комірках пам’яті, але обробляються компілятором і для програміста високого рівня непомітні); велика програма стала сукупністю процедур-підпрограм, одна з яких була головною;

2) функціональні – відсутні змінні та оператор присвоювання; програма являє собою означення функцій та правила їх використання, а виконання програми полягає в обчисленні функцій; це, як правило, інтерпретатори; прообраз в імперативних – функції, які мають обмеження (повертають лише змінні скалярного типу, а не інші функції)

**Функціональні мови.** Основна ідея, що лежить в основі функціонального програмування, - це представлення програми у вигляді математичних функцій (тобто функцій, значення яких визначається лише їх аргументами, а не контекстом виконання). Оператор присвоєння в таких мовах не використовується (або, як мінімум, його використання не заохочується). Імперативні можливості, як правило, є, але їх застосування обставлене серйозними обмеженнями. Існують мови з пасивною та з активною семантикою. Відмінність полягає, грубо кажучи, в тому, що в мовах з активною семантикою обчислення проводяться в тому ж місці, де вони описані, а у разі пасивної семантики обчислення проводиться тільки тоді, коли воно дійсно необхідне. Перші мови мають ефективнішу реалізацію, тоді як другі - кращу семантику. З мов з активною семантикою згадаємо ML і два його сучасних діалекту - Standard ML (SML) і CAML. Останній має об'єктно-орієнтованого нащадка - Objective CAML (O’CaML). Серед мов з пасивною семантикою найбільш поширені два: Haskell і його більш простий діалект Clean.

**Мови логічного програмування.** Програми на мовах логічного програмування виражені як формули математичної логіки, а компілятор намагається отримати наслідки з них. Родоначальником більшості мов логічного програмування є мова ***Prolog*** (1971). У ній є низка нащадків - ***Parlog*** (1983, орієнтований на паралельні обчислення), ***Delta Prolog*** та ін.



**Транслятор. Компілятор. Інтерпретатор**.

**Трансля́тор** (англ. *translator*) — програма або технічний засіб, який виконує перетворення чи іншу обробку [текстів програм](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%87%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4).

Транслятори поділяються на:

* [компілятори](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) — перетворюють текст програми [мовою високого рівня](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B8_%D0%B2%D0%B8%D1%81%D0%BE%D0%BA%D0%BE%D0%B3%D0%BE_%D1%80%D1%96%D0%B2%D0%BD%D1%8F) на [об'єктний код](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) чи [байт-код](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4).
* [декомпілятори](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) — навпаки, намагаються з машинного коду отримати [початковий код](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%87%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) на [високорівневій мові](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B8%D1%81%D0%BE%D0%BA%D0%BE%D0%B3%D0%BE_%D1%80%D1%96%D0%B2%D0%BD%D1%8F).
* [асемблери](https://uk.wikipedia.org/wiki/%D0%90%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80) — перетворюють текст програми [мовою асемблера](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%B0%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0) на [машинний код](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4).
* [дизасемблери](https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D0%B7%D0%B0%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80) — навпаки, намагаються розшифрувати машинний код.
* [інтерпретатори](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) — отримують текст програми та набір вхідних даних, і повертають результат виконання програми над вхідними даними.
* [препроцесори](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80) — отримують текст програми, і повертають перетворений певним чином текст програми.

Оскільки компілятори та інтерпретатори реалізують мови програмування, вони мають спільні риси: їх структура досить схожа, в основу їх реалізації покладено спільні теоретичні результати та практичні методи реалізації.

Два основні різновиди трансляторів**: компілятори** та **інтерпретатори.**

***Компілятори***спочатку повністю перекладають весь текст програми з мови високого рівня мовою машин­них команд, щоб потім можна було запускати отриману в результаті цього машинну про­граму.

***Інтерпретатори*** натомість читають текст програми мовою високого рівня та виконують його по мірі прочитання. Переклад програми на машинну мову не запам’ятовується, отже, щоб виконати ту ж програму вдруге, її потрібно знов пропустити через інтерпретатор.

Як правило, програми компілятори та інтерпретатори називаються так само, як і мови, для перекладу з яких вони призначені. Слова Паскаль, Бейсік, Сі можна сприймати і як назви мов, і як назви відповідних програм-трансляторів.

Якщо програма написана на інтерпретуючій мові, то інтерпретатор безпосередньо виконує її текст без попереднього перекладу. При цьому програма залишається на вихідній мові і не може бути запущена без інтерпретатора.

Можна сказати, що процесор комп’ютера — це інтерпретатор машинного кода.

Відповідно, мови програмування діляться на два класи — компілюючі та інтерпретуючі.

Програма на компілюючій мові за допомогою спеціальної програми компілятора перетворюється у набір інструкцій для даного типу процесора (машинний код) і далі записується у виконуємий файл (файли з розширення com, exe), котрий може бути запущеним на виконання як окрема програма. Іншими словами, компілятор перекладає програму з мови високого рівня на низькорівневу мову, зрозумілу процесору. Коротко кажучи, компілятор перекладає програму на машинну мову одразу і цілком, створюючи при цьом окрему програму, а інтерпретатор перекладає її на машинну мову під час виконання програми.

***Література***

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
3. Зелковиц М., Шоу А., Геннон Дж. Принципы разработки программного обеспечения. — М.: Мир, 1982. — 368 с. URL: [http://computersbooks.net/index.php?id1=4&category=teoriyaprogramirovaniya& author=zelkovic-m&book=1982](http://computersbooks.net/index.php?id1=4&category=teoriyaprogramirovaniya&%20author=zelkovic-m&book=1982)
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

***Для самостійного вивчення .***

1. Системи числення.
2. Перехід з однієї системи числення в іншу

Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Контрольні запитання*.**

1. Дайте визначення мови програмування.
2. Від чого залежить вибір мови програмування?
3. Що таке мнемокод?
4. Яка мова була першою широковживаною компільованою мовою? Яка ключова ідея лежала в її основі?
5. Яка з широковживаних мов програмування має найбільший вік?
6. Визначте покоління мов програмування.
7. На чому, на вашу думку, базується поділ мов програмування за поколіннями?
8. Зробіть класифікацію мов програмування
9. В чому полягає особливість імперативних мов програмування? Які мови відносяться до цієї групи?
10. В чому різниця між компілятором та інтерпретатором?
11. Наведіть приклади компіляторів та інтерпретаторів.