Функціональне програмування

Лектор Ковалюк Тетяна Володимирівна
к.т.н., доцент
tkovalyuk@ukr.net

Огляд парадигм та мов функціонального програмування

Scheme

Clojure

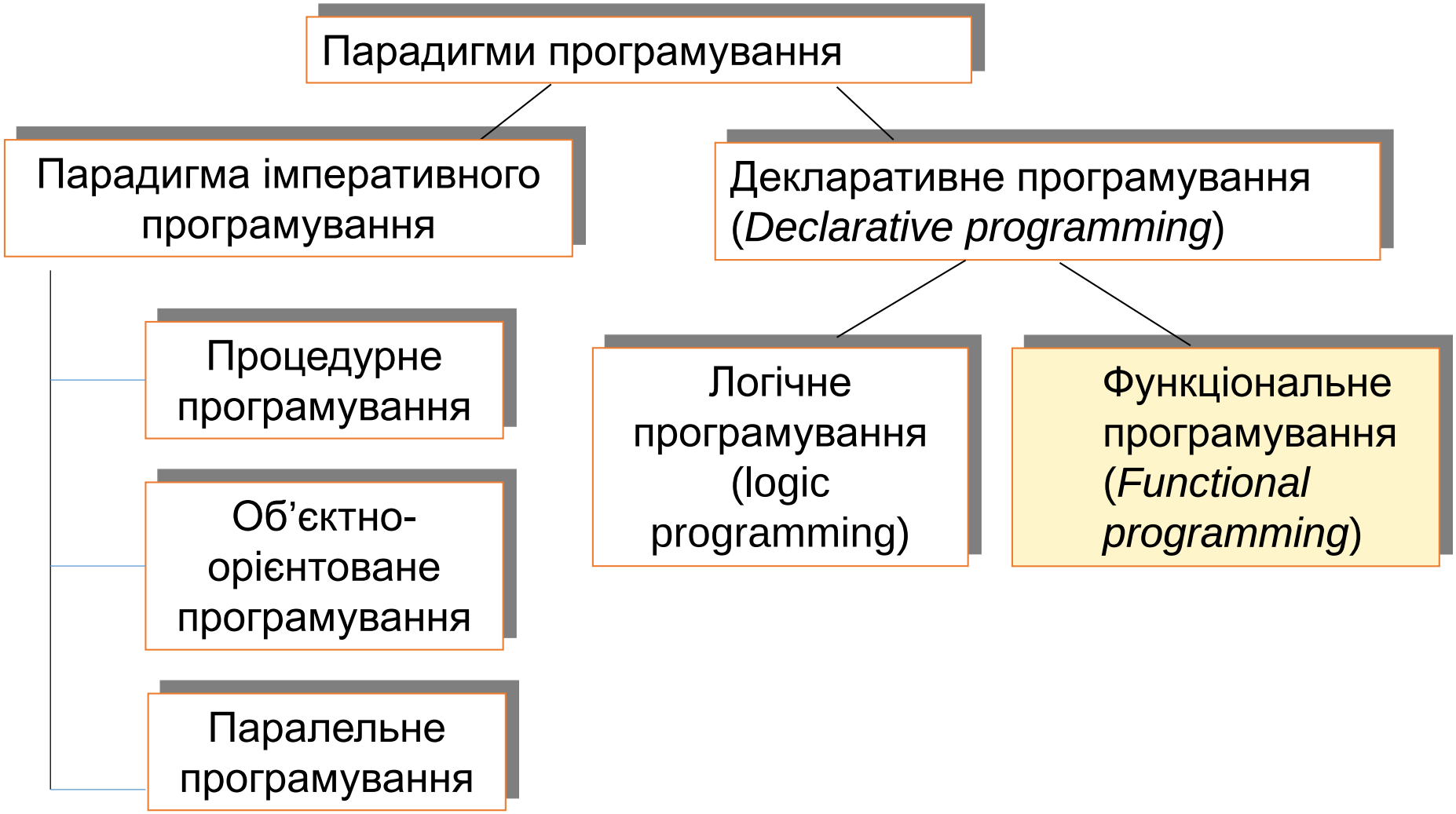
Lisp

Haskell

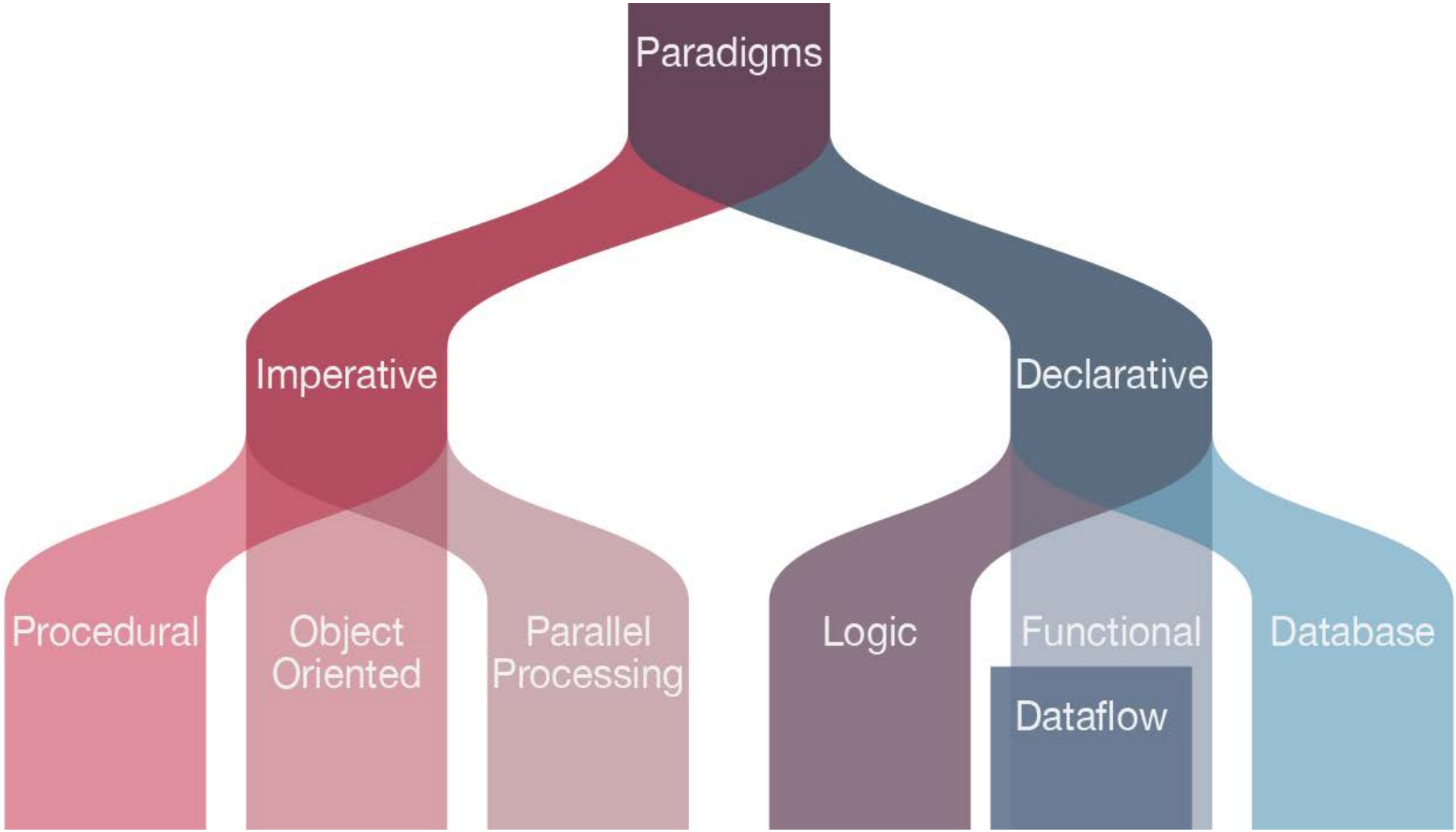
План лекції 1

1. Парадигма декларативного програмування
2. Відмінності імперативних програм від декларативних
3. Особливості функціонального програмування
4. Математичні функції як базіс функціонального програмування
5. Основні принципи функціонального програмування
6. Генеалогія та порівняння мов програмування
7. Характеристика мови LISP
8. Характеристика мови Scheme
9. Характеристика мови COMMON LISP
10. Середовища (IDE) функціонального програмування
11. Корисні посилання

Парадигми програмування



Парадигми програмування



Мови програмування

Мови програмування,
які підтримують парадигми

Алгоритмічні (процедурні) мови (**C**, **C++**, **Python Basic**, …)

Декларативні (неалгоритмічні) мови

Мови **логічного** програмування (**Prolog**, …)

Мови **функціонального** програмування (**Lisp**, **Scheme, Haskell, Erlang, ML, Scala, …)**

Особливості парадигм програмування

Парадигма імперативного програмування

- заснована на принципі неймановської архітектури комп'ютера
- стиль написання коду у вигляді набору послідовних інструкцій (команд) з використанням змінних.

Парадигма функціонального програмування

- заснована на математичних функціях

Декларативне програмування

Декларативне програмування — парадигма програмування відповідно до якої, програма описує результат, який необхідно отримати, замість опису послідовності дій отримання цього результату.

Приклад:
Веб-сторінки HTML — декларативні, оскільки вони описують, *що* містить сторінка та *що* має відображатись, наприклад:

- заголовок,
- шрифт,
- текст,
- зображення,

але *не містить інструкцій, як* її слід відображати.

Поняття декларативного програмування

- Декларативне програмування - це програмна парадигма, яка виражає логіку обчислення, не описуючи його потік контролю.
- Багато мов, що застосовують цей стиль, намагаються мінімізувати або усунути побічні ефекти, описуючи те, чого програма повинна досягти, замість того, щоб описати, як цього досягти.
- Декларативне програмування часто розглядає
  - програми - як теорії формальної логіки,
  - обчислення - як виводи в тому логічному просторі.

Декларативне програмування

Інше визначення:

Програма «декларативна», якщо її написано винятково
- функціональною мовою програмування,
- логічною мовою програмування,
- або мовою обмежень.

Декларативні мови називають мовами **надвисокого або найвищого рівня**, оскільки вони дуже близькі до людської мови і людського мислення.

Галузі застосування декларативних мов програмування

- Системи штучного інтелекту
- Автоматичний доказ теорем
- Експертні системи та оболонки експертних систем
- Системи підтримки прийняття рішень
- Системи обробки природної мови
- Планування дій роботів
- Автоматичне переведення, обробка текстів природної мови,
- САПР,
- Data-mining системи,
- Автоматичне управління,
- Бази знань,
- Символьні обчислення тощо

Стиль подання декларативних програм

1. Програма є сукупністю тверджень, що описують фрагмент предметної області або ситуацію, що склалася;
2. Описується результат або його властивості, а не методи його досягнення.
3. Програмуючи в декларативному стилі, програміст повинен описати, що треба вирішувати.
4. В основі декларативних мов лежить формалізована людська логіка.
5. Людина лише описує задачу, яку слід розв’язати, а пошуком рішення займається імперативна система програмування.

Відмінності імперативних програм від декларативних

Імперативні програми	Декларативні програми
Для отримання результатів явно конкретизують **алгоритм**	Для отримання результатів явно конкретизують **мету**
Повільніший темп розробки програм	Значно **більша** швидкість розробки застосувань
Більший за розміром програмний код	Значно **менший** розмір початкового коду
Труднощі із обробкою знань	Легкість запису знань на декларативних мовах
Іноді важкість розуміння програми	Зрозуміліші, в порівнянні з імперативними мовами, програми
Наприклад	
Алгоритм Дейкстри пошуку мінімального маршруту конкретизує послідовність дій та умов для отримання результату	Інструкція **select SQL** конкретизує властивості даних, які слід отримати від бази даних, але не процес отримання цих даних.

Парадигма Функціонального програмування

Функціональне програмування — парадигма програмування, яка розглядає **програму як обчислення математичних функцій та уникає стани та змінні дані**.

Функціональне програмування наголошує на застосуванні функцій, на відміну від імперативного програмування, яке наголошує на змінах в стані та виконанні послідовностей команд.

Особливості Функціонального програмування

Функціональне програмування є способом створення програм, в яких:

- дія - це виклик функції,
- спосіб розбиття програми - це створення нового імені функції та завдання для цього імені виразу, що обчислює значення функції,
- правило композиції - це оператор суперпозиції функцій.

Жодних комірок пам'яті, операторів присвоєння, циклів, блок схем і передачі управління.

Деталізація функціонального стилю програмування

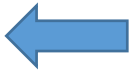
1. Функціональна програма є просто виразом, а виконання програми - процесом його обчислення.
2. Вираз відповідає математичній функції.
3. Функціями вважаються і самі числа.
4. Функції можуть передаватися в інші функції як аргументи і повертатися як результати.
5. Функція може розглядатися як структура даних, яка може бути аргументом іншої функції, і яка може повертатися як результат обчислення функції.
6. Функції можуть застосовуватися в обчисленнях, як операнди виразів.
7. Замість послідовного виконання операторів і використання циклів, функціональні мови програмування використовують рекурсивні функції, тобто функції, визначені в термінах самих себе.

Деталізація функціонального стилю програмування

8. Значення функції залежить тільки від її параметрів, та не залежить від попередніх обчислень .
9. Математичною моделлю функціональних мов є лямбда-числення.
10. Функціональне рішення є фактично формулюванням самої задачі, а не рецептом її рішення, тобто функціональна програма є специфікацією того, що потрібно зробити, а не послідовністю інструкцій, що описують, як це зробити.
11. Функціональне програмування не має поняття змінних та оператора присвоєння.
12. Відсутність змінних та оператора присвоєння є причиною **неможливості** циклів. Замість циклів використовується рекурсія.
13. Через відсутність змінних та оператора присвоєння **відсутнє** поняття стану функції.

Приклад коду імперативним і функціональним стилями

```
function sumArray (values)
{  var sum=0;
    for(var i=0;i<values.length;i++)
     { sum+=values[i];
     }
return sum;
}
sumArray([2, 5, 8]);
```



Імперативний стиль

Функціональний стиль

```
function sumArray (values)  {
  return values.reduce (
      ( previousValue,currentValue)=>previousValue+currentValue,
        0
      );
}
sumArray([2, 5, 8]);
```

Приклад функціонального програмування

Функціїя обчислення факторіалу.

Імперативна програма на мові С через цикл:

```
int  fact(int  n)
{  int  x =  1;
    while  (n >  0)
     {  x *= n; n --; }
  return  x;
}
```

Функціональна програма на мові ML через рекурсивну функцію:

```
let  rec  fact  n = if  n = 0  then  1
                else  n  *  fact(n -  1);
```

Мови Функціонального програмування

До відомих функціональних мов програмування, які використовуються в промисловості та комерційному програмуванні належить:

- Lisp
- HASKELL
- ML
- Scheme
- Erlang (паралельні програми),
- R (статистика),
- Mathematica (символьні обчислення),
- J та K (фінансовий аналіз),
- XSLT (спеціалізована мова програмування)
- APL,
- ……

Переваги функціональних мов програмування

- Програма є безліччю обчислюваних функцій
- Відсутня операція присвоєння
- Порядок виконання програми (обчислень) несуттєвий
- Швидкодія збільшена (в порівнянні з аналогічними імперативними обчисленнями)
- Компактність коду (особливо для списків і варіантних типів)
- Суперпозиції/склеювання функцій: з простих - складні, редукція
- Суперпозиція моделей: склеювання програм, ледачі обчислення
- Чисельне диференціювання/інтеграція (в т.ч. модульне)
- Алгоритми штучного інтелекту (евристичний пошук і т.п.)
- Розпаралелювання обчислень (Haskell)
- Інтерактивне налагодження, Unit -тестування
- Зіставлення із зразком

Математичні функції як базіс функціонального програмування

Поняття математичної функції.

- Математична функція - це відображення (mapping) елементів однієї множини, області визначення, в іншу множину – область значень.
- Відображення описується виразом або таблицею.
- Функції часто застосовуються до окремого елементу множини визначення, яка може бути результатом перетину декількох множин.
- Функція повертає, елемент з множини значень.

Властивості математичних функції

1. Порядок обчислення виразів, які задають відображення, управляється рекурсивними і умовними виразами, а не ітеративним повторенням і послідовністю виконання операцій, як в імперативних мовах програмування.
2. Функції завжди визначають одне і те саме значення при заданому наборі аргументів.
3. Математична функція визначає значення, а не вказує послідовність операцій над числами, що зберігаються в комірках пам'яті, для обчислення деякого значення. Тут немає змінних, як в імперативних мовах програмування, тому не може бути побічних ефектів.

Поняття простих функції

1. Визначення функцій часто записуються у вигляді імені функції, за яким слідує список параметрів в дужках і вираз, що задає відображення.
2. Нехай є функція cube (*x*) =**x***x**x*. Параметр *x* може бути будь-яким елементом з області визначення, але під час обчислення виразу, що визначається функцією, він являє собою лише один конкретний елемент.
3. Елемент, що є значенням функції, визначають шляхом обчислення виразу, що задає відображення, стосовно елементу з області визначення, заміненого значенням параметра. Наприклад, cube (2.0) в результаті дає 8.0.
4. Метод визначення **безіменних функцій** реалізується за допомогою **лямбда-числення**. Наприклад, є вираз λ (*x*) *x* * *x* * *x*. Застосування лямбда-виразу до заданого параметра записується λ (*x*) *x* * *x* * *x*(2) і повертає значення 8.

Функціональні форми

Функції вищого порядку, або функціональні форми, відрізняються тим, що вони

- або отримують функції у вигляді параметрів,
- або якась функція є результатом їх роботи,
- або має місце і те, і інше.

Найбільш поширеним видом функціональних форм є композиція функцій, параметрами якої є дві функції.

Результат композиції функцій - це функція, значення якої є, результатом застосування першої функції-параметра до результату роботи другої функції-параметра.

Наприклад, якщо

$f(x) = x + 2,$

g(x)=3*x,

то h(x)=f(g(x)) або h(x)=(3*x)+2

Основні принципи функціонального програмування

1. Усі функції - чисті

Усі функції є чистими, якщо вони задовольняють двом правилам:

1. Функція, що викликається від одних і тих самих аргументів, завжди повертає однакове значення.
2. Під час виконання функції не виникають побічні ефекти.

Перше правило:

Якщо, наприклад, викликається функція sum(2, 3), то очікується, що результат завжди буде дорівнює 5. Як тільки викликається інша функція, або звертаєтеся до змінної, що не визначена в функції, чистота функції порушується, а це в функціональному програмуванні **неприпустимо**.

Друге правило:

Побічний ефект - це зміна чогось відмінного від функції, яка виконується в поточний момент. Зміна змінної поза функцією, виведення в консоль, виклик винятку, читання даних з файлу - все це приклади побічних ефектів, які позбавляють функцію чистоти. Якщо виклик функції не змінить нічого "зовні", то можна використовувати цю функцію в будь-якому сценарії. Це відкриває дорогу конкурентному програмування і багатопотоковим додатків.

Основні принципи функціонального програмування

1. Усі функції - чисті

```
var z = 10;
function add(x, y) {
 return x + y;
 }
```

При одних й тих самих значеннях x, y функція повертатиме однаковий результат.

```
const x = 10;
const z = 10;
add (x, z); // поверне 20
```

При одних й тих самих значеннях x, y функція повертатиме однаковий результат.

```
var z = 10;
function add(x, z) {
 return x + z;
 }
```

z доступна для зміни значення, значень функція add() не є чистою, бо може видати непередбачений результат.

Основні принципи функціонального програмування

2. Усі функції є першого класу і вищого порядку (композиція функцій)

Для того, щоб функція була першокласною, у неї має бути можливість бути оголошеної у вигляді **змінної**. Це дозволяє управляти функцією як звичайним типом даних і в той же час виконувати її.

Функції вищого порядку визначаються **як функції, які використовують іншу функцію** як аргумент або повертають функцію.

```
function addOne(x) {
 return x + 1;
}
function timesTwo(x) {
return x * 2;
}
console.log(addOne(timesTwo(3))); // выведет 7
console.log(timesTwo(addOne(3))); // выведет 8
```

3. Змінні не змінюються

У функціональному програмуванні не можна змінити змінну після її ініціалізації.
Можна створювати нові, але не можна змінювати існуючі - і завдяки цьому можна бути впевненим, що жодна змінна не зміниться.

Основні принципи функціонального програмування

Запобігання побочним ефектам

Неприпустимі цикли через зміну параметрів циклів. Щоб запобігти побічним ефектам для створення циклів використовується рекурсія

Нефункціональний код

```
var acc = 0;
for (var i = 1; i <= 10; ++i) {
    acc += i;
}
console.log(acc); // выведет 55
```

Функціональний код

```
function sumRange(start, end, acc) {
    if (start > end) {
    return acc;
    }
    else {
      return sumRange(start + 1, end, acc + start);
    }
}
console.log(sumRange(1, 10, 0)); // выведет 55
```

Основні принципи функціонального програмування

4. Відносна прозорість функцій

Якщо можна замінити виклик функції на значення, що повертається, і стан при цьому не зміниться, то функція є відносно прозорою.
приклад:

Приклад прозорої функції:

```
public int addNumbers(){
  return 3 + 5;
}

addNumbers() ; // 8
```

Очевидно, що будь-який виклик цієї функції можна замінити на 8 - значить, функція відносно прозора.

Приклад непрозорої функції:

```
public void printText(){
  System.out.println("Hello World");
}

printText();
```

Ця функція нічого не повертає, але друкує текст, і при заміні виклику функції на функцію без парамментів стан консолі буде іншим.

Основні принципи функціонального програмування

5. Функціональне програмування засноване на лямбда-численні

Функціональне програмування спирається на математичну систему, що називається **лямбда-численням**:

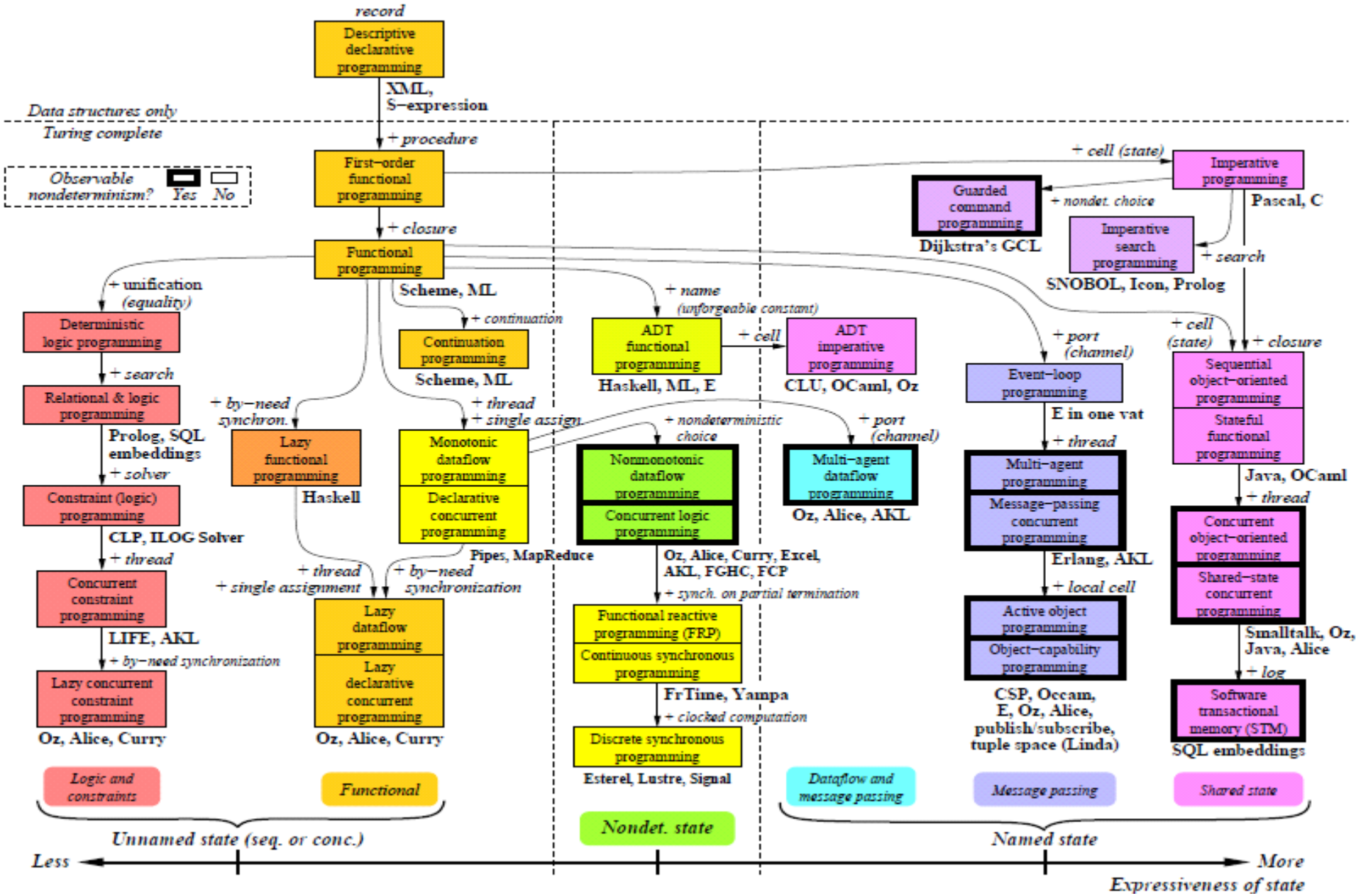
1. В лямбда-численні всі функції можуть бути **анонімними**, оскільки єдина значуща частина заголовка функції - це список аргументів.

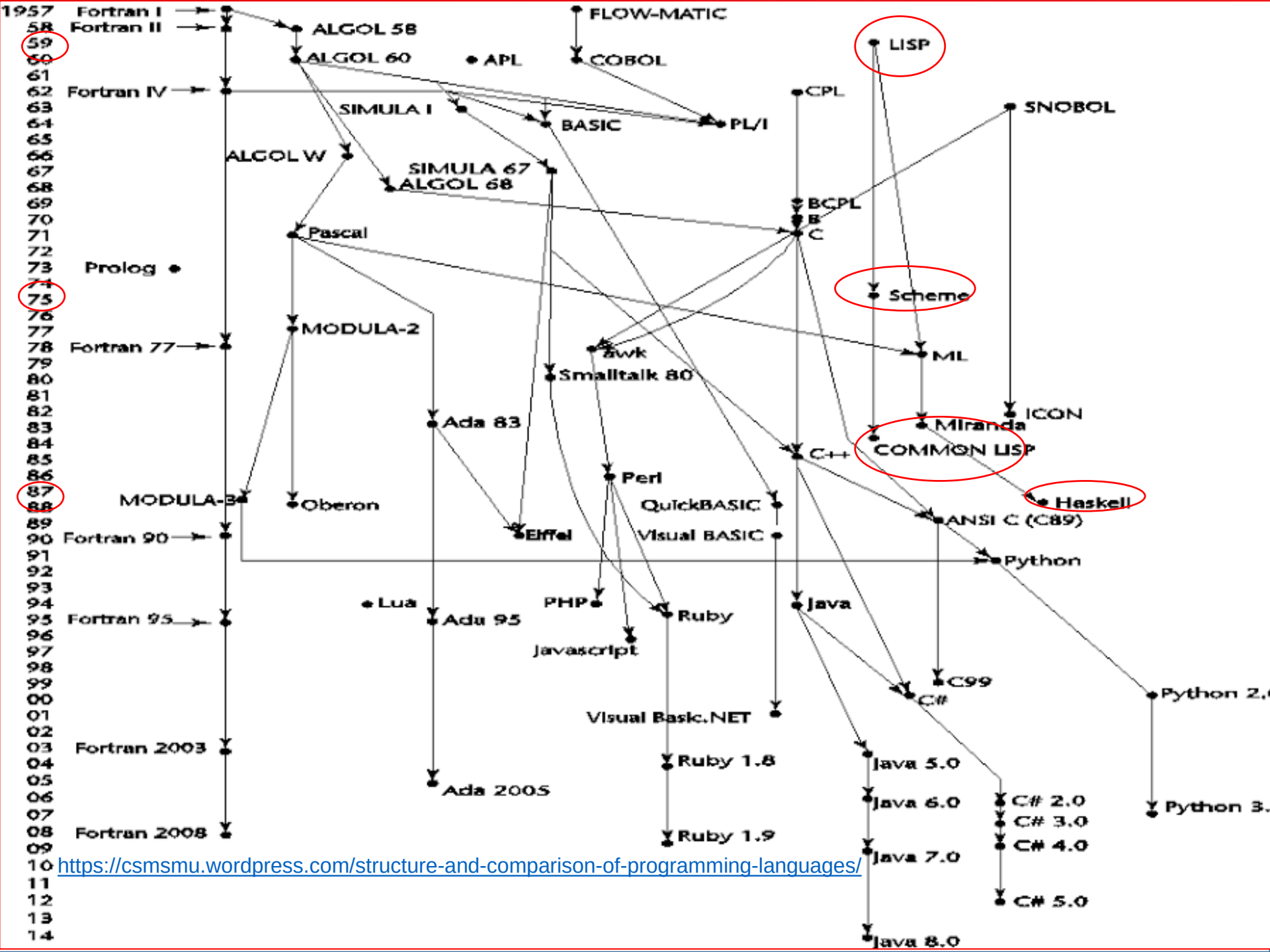
2. Під час виклику всі функції проходять процес каррінгу.

Каррінг це спосіб конструювання функцій, що дозволяє часткове застосування аргументів функції. Це означає, що можна передати всі аргументи, очікувані функцією і отримати результат, або передати частину цих аргументів і отримати назад функцію, яка очікує інші аргументи.

Цей процес **рекурсивний** і триває до тих пір, поки не будуть застосовані всі аргументи, повертаючи фінальний результат.

Генеалогія та порівняння мов програмування





https://csmsmu.wordpress.com/structure-and-comparison-of-programming-languages/

Сім'я LISP

Сім'я Lisp включає такі **діалекти**:

1. Clojure,
2. Common Lisp,
3. Dylan,
4. Erlang
5. Emacs Lisp,
6. Little b,
7. Logo,
8. Scheme,
9. Racket (також відомий як PLT Scheme),
10. Tea
11. ….

Таблиця порівнянь **реалізацій функціональних мов**
https://ru.wikipedia.org/wiki/Common_Lisp

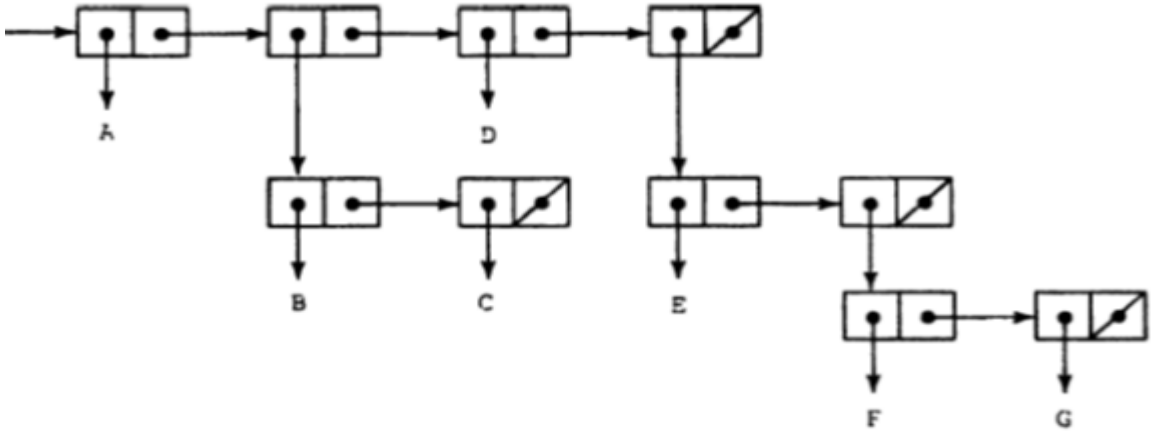
Функціональні мови програмування

https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D1%96%D1%8F:%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%BC%D0%BE%D0%B2%D0%B8_%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

Характеристика мови LISP

Типи і структури даних

- Існують тільки два типи даних в мові LISP: атоми і списки.
- Атоми, що мають вид ідентифікаторів, - це символи мови LISP.
- Числові константи також розглядаються як атоми.
- Списки зберігаються у вигляді однозв''язних структур списків, в яких кожен вузол має два покажчика і являє собою елемент списку. Перший покажчик у вузлі, що представляє собою атом, посилається на символ або числове значення. Другий покажчик у вузлі посилається на наступний елемент списку.



Характеристика мови LISP

Типи і структури даних

- Списки вказуються у вигляді розділених між собою елементів, укладених в дужки. Елементи простих списків є атомами, наприклад: (A B C D). Вкладені структури списків також вказуються за допомогою дужок.

  Наприклад, список (A (B C) D (E (F G))) являє собою список з чотирьох елементів. Перший елемент - атом A; другий елемент подсписок (B C); третій елемент-атом D: четвертий елемент-подсписок (E (F G)), який містить в якості свого другого елементу подсписок (F G).
- Для визначення функцій була обрана система лямбда-позначень

  (ім'я функції (LAMBDA (аргумент_1 ... аргументу) вираз))
- Виклики функцій записувалися в префиксной формі списку, що отримала назву польського запису:

  (Імя_функциі аргумент_1 ... аргументу)

  Наприклад, якщо + являє собою функцію, яка одержує два числових параметра, то результатом виразу (+5 7) є число 12.

Характеристика мови LISP

- Функції мови LISP викликаються за допомогою S-виразів, що представляють собою форму записи символьних виразів.
- В результаті усі структури мови LISP, **і дані, і код,** стали називатися S-виразами.
- S-вираз може бути або списком, або атомом.

Характеристика мови Scheme

- Мова Scheme, що є діалектом мови LISP, проста і популярна мова функціонального програмування, а інтерпретатори мови Scheme легкодоступні для будь-якого типу комп'ютерів.
- Scheme використовує виключно статичний огляд даних і обробляє функції як сутності першого класу.
- Функції в мові Scheme можуть бути **значеннями виразів і елементами списків**, а також можуть присвоюватися змінним і передаватися як параметри.
- Інтерпретатор мови Scheme є нескінченний цикл типу

  прочитати- обчислити-записати.
- Вирази, які викликають елементарні функції, обчислюються наступним чином: спочатку обчислюється кожен з параметрів виразу незалежно від порядку їх слідування. Потім елементарна функція застосовується до значень параметрів, і відображається результат.
- Мова Scheme **містить елементарні функції** для виконання основних арифметичних операцій. До них відносяться **+, -, * і /** для додавання, віднімання, множення і ділення.

Введення в мову Scheme

- **Списки** є основні структури даних в мові Scheme.
- Існують дві елементарні функції вибору елементів зі списків в мові Scheme: **CAR і CDR** Функція **CAR** повертає перший елемент заданого списку. Функція **CDR** повертає залишок заданого списку
- Функція **CONS** - елементарний конструктор списку. Вона створює список з двох своїх аргументів, перший з яких може бути або атомом, або списком; другий зазвичай є списком.
- Існують **предикатні функції**, яка повертає булевское значення, для порівняння списків, визначення порожнечі і наявності списку

  EQ ?, NULL? і LIST ?
- Для числових даних є предикатні функції: **=, <>, >, <...**
- Для побудови функцій використовується систему лямбда-позначень:

  (LAMBDA (L) (CAR (CDR L)))
- Функція **DEFINE** призначена для зв'язування імені зі значенням і з лямбда-виразом.

  (DEFINE (square number) (* number number))
- Як тільки інтерпретатор вирахує цю функцію, її можна використовувати, наприклад: **(square 5)**

Приклад функції на мові Scheme

Порівняння двох списків загального вигляду

```
(DEFINE (equal lisl lis2)
     (COND
          ((NOT (LIST? lisl)) (EQ? lisl lis2))
          ((NOT (LIST? lis2)) ' ())
          ((NULL? lisl) (NULL? lis2))
          ((NULL? lis2)) ’ ()
          ((equal (CAR lisl) (CAR lis2))
                    (equal (CDR lisl) (CDR lis2))
          (ELSE - ())
) )
```

Характеристика мови COMMON LISP

- Мова COMMON LISP покликана об'єднати властивості декількох діалектів мови LISP, створених на початку 1980-х років, включаючи мову Scheme, в єдину мову.
- Будучи комбінацією цих властивостей, мова COMMON LISP досить велика і складна. ЇЇ основа - це вихідна мова LISP.
- Список властивостей мови COMMON LISP:
  - велика кількість типів і структур даних, включаючи записи, масиви, комплексні числа і символьні рядки;
  - потужні операції введення і виведення;
  - пакети для об'єднання в модулі наборів функцій і даних, а також управління доступом;
  - імперативні властивості мови Scheme

✓ Мови Scheme і COMMON LISP протилежні одна одній.
✓ Мова Scheme набагато менша за розмірами і ясніша, частково завдяки тому, що використовує виключно статичні області видимості.
✓ Мова COMMON LISP призначалася для комерційного використання і згодом набула широкого поширення в додатках, пов'язаних зі

Середовища (IDE) функціонального програмування

IDE	Мови ФП	Сайт
GNU Common Lisp (GCL)	Common Lisp	http://www.tucows.com/preview/7932/GCL https://www.cs.utexas.edu/users/novak/gclwin.html
Haskell Platform for Windows **8.6.5**.	**Haskell**	https://www.haskell.org/platform/windows.html
Glasgow Haskell Compiler	**Haskell**	https://www.haskell.org/ghc/
Haskell Stack	**Haskell**	https://tech.fpcomplete.com/haskell/get-started/windows
Erlang/OTP 22.0	*Erlang*	https://www.erlang.org/downloads
Compile lisp online	Lisp	https://rextester.com/l/common_lisp_online_compiler
HomeLisp Portable	Lisp	http://homelisp.ru/
LispWorks Personal Edition 6.0.1	Lisp	https://www.softportal.com/get-19673-lispworks-personal-edition.html

Середовища (IDE) функціонального програмування

IDE	Мови ФП	Сайт
Online Scheme compiler	Scheme	https://repl.it/languages/scheme
LispIDE	Corman Common Lisp, Steel Bank Common Lisp, CLISP, Gnu Common Lisp, Gambit Scheme, Bigloo Scheme, SCM Scheme, Arc, newLISP, OpenLisp, Clozure Common Lisp, PicoLisp, Clojure, Armed Bear Common Lisp	https://www.daansystems.com/lispide/
Dr.Racket/Dr.Scheme	Racket Scheme	https://download.racket-lang.org/ https://racket-lang.org/ (R6RS standard)

Література з декларативного програмування

https://github.com/tkovalyuk/

Обзор литературы о функциональном программировании.
http://alexott.net/ru/fp/books/
Обзор литературы о функциональном программировании
http://fprog.ru/2009/issue1/alex-ott-literature-overview/

Programming paradigms for dummies: what every programmer should know
https://blog.acolyer.org/2019/01/25/programming-paradigms-for-dummies-what-every-programmer-should-know/

Основные принципы программирования: функциональное программирование
https://tproger.ru/translations/functional-programming-concepts/

Дякую за увагу

Доц. кафедри ПСТ,
к.т.н. Ковалюк Т.В.
tkovalyuk@ukr.net