

PEDAGOGIK MAHORAT

11
2023

ISSN 2181-6833



9 772181 683 301

ПЕДАГОГИЧЕСКОЕ МАСТЕРСТВО

Научно-теоретический и методический журнал

№ 11, 2023

Решением Высшей аттестационной комиссии при Кабинете Министров Республики Узбекистан от 29 декабря 2016 года журнал включён в перечень изданий, рекомендованных для публикации научных результатов статей по направлениям «Педагогика» и «Психология».

Журнал основан в 2001 году.

Журнал выходит 12 раз в год.

Журнал зарегистрирован Бухарским управлением агентства по печати и массовой коммуникации Узбекистана.

Свидетельство о регистрации средства массовой информации № 05-072 от 22 февраля 2016 г.

Учредитель: Бухарский государственный университет

Адрес редакции: 200117, Узбекистан, г. Бухара, ул. Мухаммад Икбол, 11.

E-mail: nashriyot_buxdu@buxdu.uz

РЕДАКЦИОННАЯ КОЛЛЕГИЯ:

Главный редактор: Адизов Бахтиёр Рахманович – доктор педагогических наук, профессор

Ответственный редактор: Сайфуллаева Нигора Закиралиевна – доктор философии педагогических наук (PhD)

Хамидов Обиджон Хафизович, доктор экономических наук

Бегимкулов Узакбай Шаимкулович, доктор педагогических наук, профессор

Навроз-заде Бахтиёр Нигматович, доктор экономических наук, профессор

Махмудов Мэлс Хасанович, доктор педагогических наук, профессор

Ибрагимов Холбой Ибрагимович, доктор педагогических наук, профессор

Расулов Тулкин Хусенович, доктор физико-математических наук, профессор

Янакиева Елка Кирилова, доктор педагогических наук, профессор (Болгария)

Андрюченко Елена Васильевна (Институт физико-математического, информационного и технологического образования НГПУ, Новосибирск, Россия)

Ромм Татьяна Александровна (Институт истории, гуманитарного, социального образования ФГБОУ ВО НГПУ, Новосибирск, Россия)

Чудакова Вера Петровна, кандидат психологических наук (Национальная академия педагогических наук Украины, Украина)

Хамроев Алижон Рузикулович, доктор педагогических наук (DSc), доцент

Каххаров Сиддик Каххарович, доктор педагогических наук, профессор

Махмудова Муяссар, доктор педагогических наук, профессор

Козлов Владимир Васильевич, доктор психологических наук, профессор (Ярославль, Россия)

Таджиходжаев Закирходжа Абдусаттарович, доктор технических наук, профессор

Аманов Мухтор Рахматович, доктор технических наук, профессор

Ураева Дармоний Саиджановна, доктор филологических наук, профессор

Дурдиев Дурдимурад Каландарович, доктор физико-математических наук, профессор

Махмудов Насыр Махмудович, доктор экономических наук, профессор

Олимов Ширинбой Шарофович, доктор педагогических наук, профессор

Чариев Иргаш Тураевич, доктор педагогических наук, профессор

Киямов Нишон Содикович, доктор педагогических наук, профессор

Шомирзаев Махматмурод Хурамович, доктор педагогических наук, профессор

Рузиева Дилноза Исомжоновна, доктор педагогических наук, профессор

Курбонова Гулноз Несматовна, доктор педагогических наук (DSc)00

Тухсанов Кахрамон Рахимбоевич, доктор филологических наук, доцент

Назаров Акмал Мардонович, доктор философии психологических наук (PhD), доцент

Жумаев Рустам Ганиевич, доктор философии политических наук (PhD), доцент

Нуруллоев Фируз Нумонжонович, доктор философии педагогических наук (PhD)

Навроз-заде Лайли Бахтиёровна, доктор философии экономических наук (PhD)

21.	NURMATOV G‘ayrat Mustofoqulovich	Boshlang‘ich ta‘lim o‘qituvchilarining kasbiy prognostik kompetensiyalarini rivojlantirish shart-sharoitlari	109
22.	NURUMBEOVA Yarkinay Anarmatovna	Bo‘lajak mutaxassislarni kouching kompetensiyasini shakllantirishdagi asosiy komponentlar	115
23.	OLIMOV Laziz Yarashovich	Sud-psixolog eksperti kompetentligining o‘ziga xosligi	120
24.	OTAYEVA Salamat Sabirovna	Steam ta‘limi texnologiyasi asosida ilmiy qarashlarni tarkib toptirish	127
25.	QODIROV Ikrom Davronovich	Shaxsga yo‘naltirilgan ta‘lim texnologiyalari asosida tarix darslarini samarali tashkil qilish usullari	132
26.	QURBONOV Shuhrat Zarifovich	Steam fanlarni modellashtirishda dasturlar majmuasining afzalliklari	136
27.	OLIMOV Shirinboy Sharofovich, SAMIYEVA Shaxnoz Xikmatovna	Talabalarda kreativlik va yangicha dunyoqarashni rivojlantirish	140
28.	SAYLIYEVA Gulrux Rustam qizi	Talabalarining o‘qitilayotgan fanlarga qiziqishini oshirishda foydalaniladigan samarali pedagogik metodlar	144
29.	TURDIMURODOV Baxtiyor Qurbonovich	Kasb ta‘limi o‘qituvchilarining laboratoriya mashg‘ulotlarini o‘rganishda kreativlik sifatlarini rivojlantirishning samarali yo‘llari	148
30.	UMAROV Lutfillo Murodilloyevich	Uzluksiz ta‘lim yaxlit makonini modellashtirishda integrativ yondashuvni amalga oshirishning nazariy tahlili	154
31.	USMONOV Abdurashid Norbo‘tayevich	Malaka oshirish jarayonida o‘qituvchilar bilan olib boriladigan faoliyat mazmuniga doir tadqiqotlar	158
32.	UTAMURADOV Umarjon Muxammadkulovich	O‘quvchilarda huquqiy kompetensiyalarni shakllantirishning yo‘naltiruvchi modeli	163
33.	YO‘LDOSHEV Sardorbek Asliddin o‘gli	Xorij olimlari tomonidan sibling munosabatlarini o‘rganishda ijtimoiy-psixologik jihatlarining ahamiyati	169
34.	ZARMASOV Sherzod Raximberdiyevich	Kasbiy refleksiya asosida bo‘lajak o‘qituvchilarning psixologik tayyorgarligini rivojlantirish	174
35.	АТАЕВА Гульсина Исроиловна	Парадигмы программирования и способы обучения программированию	179
36.	АХМЕДОВА Дилноза Эшназар қизи	Масофали ўқитиш воситасида талабаларнинг аудиториядан ташқари мустақил таълимини ташкил этишнинг методикаси	186
37.	КАМОЛОВА Ширин Ўсаровна	Талабалар илмий дунёқарабини юксалтиришининг ахборотли таъминотининг педагогик самарадорлиги	189
38.	НУРАЛИЕВА Парвина Эркиновна	Цифровые компетенции как основа трансформации профессионального образования	195
39.	САМАРОВА Шохиста Рабиджановна	Педагог фаолияти самарадорлигининг шахслилик детерминантлари	199
40.	ТОЖИБОЕВ Марат Нормадович	Индивидуал ёндашув асосида талабалар педагогик тафаккурини ривожлантириш масалалари	204
41.	NAZAROVA Manzura Mustafo qizi	O‘zbekistonda o‘qish va yashashda chet ellik talabalarining ijtimoiy-psixologik moslashuv shartlari	209
MAKTABGACHA VA BOSHLANG‘ICH TA‘LIM			
42.	BEKCHANOVA Feruza Marimboy qizi	Maktabgacha yoshdagi bolalarning ekologik madaniyatini shakllantirish	214
43.	KORAYEVA Latifa Zarifovna	Boshlang‘ich sinf bolalarida o‘quv faoliyatining xususiyatlari	220
44.	QODIROVA Malikaxon	Maktabgacha yoshdagi bola idrokini mashg‘ulotlar va	224

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ И СПОСОБЫ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

*Атаева Гульсина Исроиловна,
преподаватель Бухарского
государственного университета
g.i.ataeva@buxdu.uz*

В данной статье рассматривается тема обучения парадигмам программирования. Она охватывает основные понятия и принципы различных парадигм, таких как императивное, декларативное. В статье рассматриваются преимущества и недостатки каждой парадигмы, а также сравнивает их между собой. Статья также включает в себя примеры кода на различных языках программирования, чтобы наглядно продемонстрировать применение каждой парадигмы. В целом, статья посвящена способам обучения парадигмам программирования, предназначенные для опытных разработчиков, чтобы помочь им передать студентам свои знания и навыки в области программирования.

Ключевые слова: *способы мышления, программный код, императивное программирование, декларативное программирование, языки программирования, запись циклов на языке программирования, концепции программирования.*

DASTURLASH PARADIGMALARI VA DASTURLASHNI O'RGATISH USULLARI

Ushbu maqolada dasturlash paradigmalarini o'qitish mavzusi ko'rib chiqiladi. U imperativ, deklarativ va boshqalar kabi turli paradigmalarining asosiy tushunchalari va tamoyillarini qamrab oladi. Maqolada har bir paradigmanning afzalliklari va kamchiliklari ko'rib chiqilgan, shuningdek, ular bir-biri bilan taqqoslangan. Maqola, shuningdek, har bir paradigmanning qo'llanilishini vizual ravishda namoyish etish uchun turli xil dasturlash tillaridagi kod namunalarini o'z ichiga olgan, umuman olganda, maqola dasturlash paradigmalarini o'rgatish usullariga bag'ishlangan bo'lib, ba'zilar tajribali dasturchilar uchun talabalarga dasturlash bo'yicha bilim va ko'nikmalarini yetkazishda yordam berish uchun mo'ljallangan.

Kalit so'zlar: *fikrlash usullari, dastur kodi, imperativ dasturlash, deklarativ dasturlash, dasturlash tillari, dasturlash tilida sikllarni yozish, dasturlash tushunchalari.*

PROGRAMMING PARADIGMS AND WAYS OF TEACHING PROGRAMMING

This article discusses the topic of teaching programming paradigms. It covers the basic concepts and principles of various paradigms, such as imperative, declarative, etc. The article discusses the advantages and disadvantages of each paradigm, and compares them with each other. The article also includes code examples in various programming languages to demonstrate the application of each paradigm. In general, the article is devoted to ways of teaching programming paradigms, designed for experienced developers to help them transfer their knowledge and skills in programming to students.

Keywords: *ways of thinking, program code, imperative programming, declarative programming, programming languages, writing cycles in a programming language, programming concepts.*

Введение. Парадигмы программирования – это различные способы мышления о том, как решать проблемы с помощью кода. Они могут влиять на дизайн, структуру и стиль ваших программ, а также на инструменты и языки, которые вы используете. Изучение различных парадигм программирования может помочь вам расширить свои навыки, творческий потенциал и универсальность как программиста. Но как можно эффективно обучать парадигмам программирования? Одними из самых распространённых парадигм программирования являются императивное и декларативное программирование — это старые, широкие, полярно противоположные парадигмы, которые сегодня часто используются для описания других парадигм программирования. Они отличаются тем, что императивное программирование шаг за шагом сообщает компьютеру, как что-то сделать, в то время как декларативное программирование объявляет конечный результат и позволяет компьютеру понять, как его достичь.

При *декларативном подходе* выражается логика вычисления без отсутствия описания потока управления, тогда как в *императивной* парадигме программирования используются утверждения, изменяющие состояние программы (рисунок 1).



Рисунок 1. Описание императивного и декларативного подходов

Современные языки программирования, такие языки, как Python и C++— мультипарадигматичны, поскольку поддерживают написание кода в более чем одной парадигме. Рассмотрим некоторые способы, которые помогут вам внедрить и изучить различные парадигмы вместе с вашими студентами [4].

Обсуждение. Один из лучших способов обучения парадигмам программирования - показать примеры кода, использующего различные парадигмы. Вы можете использовать онлайн-ресурсы, такие как metanit.com, чтобы найти фрагменты кода, иллюстрирующие одну и ту же задачу на разных языках и парадигмах. Показывая примеры, вы можете помочь своим студентам увидеть различия и сходства между парадигмами, а также преимущества и недостатки каждого подхода.

Императивное программирование – это парадигма программирования, в которой программа состоит из последовательных инструкций, указывающих компьютеру, как выполнять определенную задачу. Основная идея заключается в изменении состояния программы с помощью последовательности команд [5].

Пример кода на C++ в императивном стиле:

```
```cpp
#include
int main() {
 int sum = 0;
 for (int i = 1; i <= 10; i++) {
 sum += i;
 }
 std::cout << "Сумма чисел от 1 до 10: " << sum << std::endl;
 return 0;
}
```
```

Этот пример демонстрирует использование цикла for для вычисления суммы чисел от 1 до 10.

Декларативное программирование – это парадигма программирования, в которой программа описывает желаемый результат, а не последовательность команд для его достижения. Он сосредотачивается на описании «что» должно быть сделано, а не «как».

Пример кода на Python в декларативном стиле:

```
```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = sum(numbers)
```

```
print("Сумма чисел от 1 до 10:", sum)
```

Этот пример использует встроенную функцию ‘sum()’, которая принимает список чисел в качестве аргумента и вычисляет их сумму [7].

В обоих примерах результат одинаков - сумма чисел от 1 до 10, но подходы к написанию кода различаются. В императивном стиле программист явно указывает, как выполнить задачу, в то время как в декларативном стиле программа описывает желаемый результат, и компьютер самостоятельно находит способы его достижения.

Очень часто приходится писать коды программ с ветвлениями, для этого используется конструкция **if-else** для разветвления программы на несколько путей выполнения кода в зависимости от некоторого условия.

Рассмотрим пример, переменная *a* сначала проверяется на условие  $a > 0$ . Если это условие выполняется, то выводится сообщение “a is positive”. Если это условие не выполняется, то переходит к проверке следующего условия -  $a == 0$ . Если и это условие не выполняется (т.е. значение *a* не равно нулю), то выводится сообщение “a is negative”.

Код на C++.

```
{
 int a;
 std::cout << "Enter a number: ";
 std::cin >> a;
 if (a > 0) {
 std::cout << "a is positive" << std::endl;
 } else if (a == 0) {
 std::cout << "a equals zero" << std::endl;
 } else {
 std::cout << "a is negative" << std::endl;
 }
}
```

Ниже приведен пример кода на Python:

```
a = 5
if a > 0:
 print("a is positive")
elif a == 0:
 print("a is zero")
else:
 print("a is negative")
```

Теперь рассмотрим пример с использованием оператора выбора. В этом примере программа запрашивает у пользователя число от 1 до 3, а затем использует оператор выбора для выполнения различных действий в зависимости от введенного числа.

В C++, оператор выбора **switch** используется для выполнения различных действий в зависимости от значения выражения.

Код C++:

```
{
 int choice;
 std::cout << "Введите число от 1 до 3: ";
 std::cin >> choice;
 switch (choice) {
 case 1:
 std::cout << "Вы выбрали 1" << std::endl;
 break;
 case 2:
 std::cout << "Вы выбрали 2" << std::endl;
 break;
 case 3:
 std::cout << "Вы выбрали 3" << std::endl;
 }
}
```

```
break;
default:
 std::cout << “Неверный выбор” << std::endl;
}
```

В этом примере программа запрашивает у пользователя число от 1 до 3, а затем использует оператор **switch** для выполнения различных действий в зависимости от введенного числа. Если введенное число не соответствует ни одному из **case**, выполняется код в блоке **default**.

Код Python:

```
choice = int(input(“Введите число от 1 до 3: “))
```

```
if choice == 1:
 print(“Вы выбрали 1”)
elif choice == 2:
 print(“Вы выбрали 2”)
elif choice == 3:
 print(“Вы выбрали 3”)
else:
 print(“Неверный выбор”)
```

Этот код сначала запрашивает у пользователя ввод числа. Затем он использует условные операторы **if**, **elif** и **else**, чтобы выполнить различные действия в зависимости от введенного числа. Если введенное число не соответствует ни одному из вариантов, будет выполнен блок кода в **else** [1].

При сравнении кодов, можно заметить, что запись на языке программирования Python выглядит более компактно и удобна в использовании.

Теперь рассмотрим организацию циклов на разных языках программирования.

В языках программирования C++, Java и Python используются различные подходы к организации циклов, но все они служат одной цели - повторению набора инструкций определенное количество раз или до выполнения заданного условия.

В C++ для организации циклов используются ключевые слова **“for”**, **“while”** и **“do-while”**. Например, цикл **“for”** используется для выполнения блока кода определенное количество раз:

```
for(int i = 0; i < 10; i++){
 // тело цикла
}
```

Цикл **“while”** используется, когда количество итераций заранее не известно и продолжает выполняться, пока условие истинно:

```
int a = 5;
while(a > 0){
 a--;
 //тело цикла
}
```

Наконец, цикл **“do-while”** выполняется хотя бы один раз, даже если условие ложно:

```
do{
 // тело цикла
}while(a == 42);
```

В Java циклы организованы аналогично C++, с использованием ключевых слов **“for”**, **“while”**, **“do-while”**, а также **“foreach”** для итерации по коллекциям:

```
цикл for
for(int i = 0; i < 10; i++) {
 System.out.println(i);
}
цикл while
int a = 5;
while(a > 0) {
 a--;
 System.out.println(a);
}
цикл do-while
do {
```

```
System.out.println("Hello, World!");
} while (true);
итерация по коллекции:
for(String s : collection) {
 System.out.println(s);
}
```

Python предлагает более простой и понятный синтаксис для организации циклов. Для этого используются ключевые слова **“for”**, **“while”** и **“break”**:

```
цикл for
for i in range(10):
 print(i)
цикл while
a = 5
while a > 0:
 a -= 1
 print(a)
//прерывание цикла
while True:
 n = int(input("Введите число: "))
 if n == 42:
 break
 print("Число не равно 42")
```

Все эти языки предоставляют возможность организации циклов для решения различных задач. Выбор языка и подхода к организации циклов зависит от предпочтений и опыта программиста. Проанализировав записи можно сделать вывод, что наиболее компактной записью выглядит на языке Python, однако, стоит заметить на остальных, приведённых в примере языках, запись цикла более понятна для обучающегося основам программирования. Чтобы понять, как работает цикл рекомендуется рассмотреть запись на языке C++.

Рассматривая вышеприведённые примеры, можно сказать, что циклы являются одной из основных конструкций в программировании, позволяющих повторять определенный набор инструкций несколько раз. В языках программирования C++, Java и Python существуют различные типы циклов, которые могут быть использованы в зависимости от конкретных задач.

В C++ наиболее распространенными типами циклов являются циклы **for**, **while** и **do-while**. Цикл **for** обычно используется, когда заранее известно количество итераций, которое нужно выполнить. Внутри цикла указываются начальное значение, условие продолжения и инструкция инкремента/декремента. Цикл **while** используется, когда количество итераций неизвестно, а условие продолжения или завершения цикла проверяется перед каждой итерацией. Цикл **do-while** также используется, когда количество итераций неизвестно, но условие проверяется после каждой итерации.

В Java также существуют циклы **for**, **while** и **do-while**, которые работают аналогично циклам в C++. Однако, в Java добавлено еще несколько типов циклов, таких как цикл **foreach** и циклы с метками. Цикл **foreach** используется для итерации по элементам массива или коллекции. **For-each** — это разновидность цикла **for**, которая используется, когда нужно обработать все элементы массива или коллекции. “For each” с английского так и переводится — “для каждого”. Собственно, само словосочетание **foreach** в этом цикле не используется. Циклы с метками позволяют управлять выполнением нескольких вложенных циклов, указывая метку и используя операторы **break** и **continue** для управления выполнением циклов.

В Python циклы **for** и **while** работают по принципу, схожем с циклами в C++ и Java. Цикл **for** в Python используется для итерации по элементам коллекции, а также для выполнения указанного числа итераций с помощью функции **range()**. Цикл **while** используется, когда условие продолжения или завершения цикла проверяется перед каждой итерацией. В Python отсутствует цикл **do-while**, однако его функциональность можно эмулировать с помощью цикла **while** и дополнительных проверок.

Таким образом, циклы в языках программирования C++, Java и Python имеют общие особенности, но также имеют некоторые различия, особенно в контексте дополнительных типов циклов, которые добавлены в Java и отсутствуют в Python. Важно выбрать подходящий тип цикла в зависимости от конкретной задачи и требований программы.

Пример декларативного программирования в Python:

Функции **any()** и **all()**

проверяют, удовлетворяют ли элементы объекта условию.

**any()** принимает итерируемый объект (например, список `nums`) в качестве аргумента и возвращает `True`, если хотя бы один элемент в списке считается `True`. Если все элементы ложные или `nums` пуст, то `any()` возвращает значение `False`. **all()** тоже принимает такой объект в качестве аргумента и возвращает значение `True`, если все элементы в нем считаются истинными, или если итерируемый объект пуст. Если там есть хотя бы один элемент, который считается `False`, то `all()` вернет `False`.

```
nums = [1, 3, 5, 7, 9]
print(any(x % 2 == 0 for x in nums)) # False
print(all(x % 2 != 0 for x in nums)) # True
```

Здесь мы можем заметить, что декларативное программирование позволяет скомпоновать код и использовать готовые функции без особых разъяснений «как и что». Поэтому чтобы научить программировать, лучше использовать императивный метод, который позволяет понять, что и как происходит. Декларативное программирование больше используют профессионалы.

После демонстрации примеров вы должны объяснить концепции и принципы, лежащие в основе каждой парадигмы. Вы можете использовать диаграммы, аналогии или метафоры, чтобы помочь своим студентам понять абстрактные идеи. Например, вы можете объяснить, что императивное программирование похоже на предоставление пошагового рецепта повару, в то время как декларативное программирование похоже на составление функций, которые преобразуют ингредиенты в блюда. Вы также можете объяснить ключевые особенности и термины каждой парадигмы, такие как состояние, побочные эффекты, рекурсия или логический вывод [2].

Как только ваши ученики получают базовое представление о парадигмах, вы должны предоставить им возможность практиковать полученные навыки и применять их к реальным проблемам. Вы можете использовать онлайн-платформы для поиска упражнений и задач, которые проверяют различные парадигмы. Вы также можете разрабатывать свои собственные проекты или задания, которые требуют от ваших учеников использовать различные парадигмы или сравнивать их. Например, вы можете попросить их реализовать алгоритм сортировки с использованием императивного и декларативного программирования или решить головоломку с использованием логического программирования.

Заключительным шагом в обучении парадигмам программирования является поощрение размышлений и обратной связи. Вам следует попросить своих учеников поделиться своими мыслями и опытом использования различных парадигм. Вы можете использовать такие вопросы, как: «Как вы отнеслись к использованию этой парадигмы? Что вам в этом понравилось или не понравилось? Что вам показалось легким или трудным? Как это повлияло на качество вашего кода и производительность? Как это соотносилось с другими парадигмами или языками, которые вы знаете?» Размышляя, ваши ученики могут углубить свои знания и оценить различные парадигмы.

**Заключение.** Обучение парадигмам программирования – это не разовое мероприятие. Это непрерывный процесс, который требует постоянного обучения и исследований. Вы всегда должны быть в курсе последних разработок и тенденций в парадигмах программирования, а также появляющихся новых языков и инструментов. Вы также должны познакомить себя и своих студентов с различными парадигмами и языками, даже если они не имеют прямого отношения к вашим текущим проектам или целям. Поступая таким образом, вы можете расширить свой кругозор и открыть для себя новые возможности в программировании.

Статья обсуждает важность обучения парадигмам программирования для развития навыков и знаний в области ИТ. Обучение различным парадигмам помогает программистам лучше понимать и анализировать проблемы, а также разрабатывать эффективные и надежные решения.

### Литература:

1. Атаева Гульсина Исроиловна, Минич Людмила Станиславовна Создание вывода скрипта python // Вестник науки и образования. 2021. №1-2 (104). URL: <https://cyberleninka.ru/article/n/sozdanie-vyvoda-skripta-python>.

2. Атаева Гульсина Исроиловна, Адизова Зухро Маруф Кызы Конвертирование изображений в формат pdf с помощью python // Universum: технические науки. 2022. №4-1 (97). URL: <https://cyberleninka.ru/article/n/konvertirovanie-izobrazheniy-v-format-pdf-s-pomoschu-python>.
3. Атаева Г.И., Асадова О.А. Проблемы и решения в преподавании информатики // Приоритетные направления развития науки и образования. – 2021. – С. 169-171.
4. <https://medium.com/nuances-of-programming/декларативный-код-против-императивного-b640ea08835f>
5. <https://skine.ru/articles/10429/#:~:text=Императивное%20и%20декларативное%20программирование%20—,компьютеру%20понять%2C%20как%20его%20достичь>
6. <https://otus.ru/journal/imperativnoe-i-deklarativnoe-programmirovanie/>
7. <https://tproger.ru/translations/imperative-declarative-programming-concepts>