DOI 10.37882/2223–2966.2020.07.33

# РАЗРАБОТКА КОГНИТИВНО-ЭРГОНОМИЧЕСКОГО СИНТАКСИСА ДЛЯ НОВОГО АППАРАТНО-ОРИЕНТИРОВАННОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

## DEVELOPMENT OF COGNITIVE-ERGONOMIC SYNTAX FOR A NEW HARDWARE-ORIENTED PROGRAMMING LANGUAGE

***A. Tretyak***
***E. Tretyak***
***E. Vereshchagina***

*Summary.* The article considers programming languages from the point of view of ergonomics of their syntax. By using a cognitive-ergonomic approach to syntax design, you can improve the programming language according to the criteria "intelligibility of language constructions" and "simplicity of the language", which in turn increases the efficiency of the programmer. Based on the research, the structure of a new programming language is proposed.

*Keywords:* programming language development, hardware-oriented programming languages, cognitive ergonomics, development environment, programming.

***Третьяк Александр Викторович***
*Аспирант, Дальневосточный федеральный университет, г. Владивосток*
*alextretyak2@gmail.com*
***Третьяк Екатерина Викторовна***
*Специалист, Дальневосточный федеральный университет, г. Владивосток*
*tretiak.ev@dvfu.ru*
***Верещагина Елена Александровна***
*К.т.н., доцент, Дальневосточный федеральный университет, г. Владивосток*
*everesh@mail.ru*

*Аннотация.* В статье рассматриваются языки программирования с точки зрения эргономичности их синтаксиса. За счет использования когнитивно-эргономического подхода к проектированию синтаксиса можно улучшить язык программирования по критериям «понятность конструкций языка» и «простота языка», что в свою очередь повышает эффективность работы программиста. На основании проведенного исследования предлагается структура нового языка программирования.

*Ключевые слова:* разработка языка программирования, аппаратно-ориентированные языки программирования, когнитивная эргономика, среда разработки, программирование.

## Введение

Переход от программирования в машинных кодах к автокодам и ассемблерам, а затем языкам высокого уровня позволил существенно повысить производительность труда программистов. Следовательно, эффективность создания программного продукта в определенной мере зависит от языка. Исходя из этого, можно высказать предположение: при прочих равных условиях скорость решения человеческим мозгом интеллектуальных задач зависит от когнитивного качества профессионального языка, с помощью которого решаются указанные задачи [1].

Многие авторы подчеркивают, что выбор эффективного языка может оказать благотворное влияние на продуктивность мышления, а соответственно качество создаваемого продукта. Эрнст Шредер пишет, что употребление удачных знаков позволяет значительно усилить человеческое мышление. Неудачные языки даже простую проблему способны сделать неразрешимой. И наоборот, задача, получившая удобное знаковое выражение, оказывается наполовину решенной [2].

Например, замена языка римских чисел на язык арабских чисел дала возможность резко увеличить производительность труда при выполнении арифметических действий. Как отмечает Дэвид Марр, "это главная причина того, почему римская культура не смогла развить математику так, как это сделали ранние арабские культуры" [3].

Когнитивная эргономика — это новая научная дисциплина, изучающая проблему улучшения профессиональных языков и усиления естественного (а не искусственного) интеллекта. Ее задача — создать принципиально новые формы представления профессиональных знаний, максимально комфортные и удобные для работы глаза и мозга. Знания можно назвать эргономичными, если они максимально наглядны и пригодны для быстрого восприятия и усвоения. [4]

При разработке приложений программист выбирает язык программирования, ориентируясь на различные свойства языка, а также конкретную задачу.

Свойства, которыми обладают языки программирования:

- понятность (удобочитаемость) конструкций языка;
- надёжность (степень автоматического обнаружения ошибок);
- гибкость (возможности, которые язык предоставляет программисту);
- простота языка;
- мобильность (возможность переносить программы с одной платформы на другую с относительной легкостью);
- суммарная стоимость использования языка [5, стр. 9].

В рамках когнитивно-эргономического подхода можно выделить такие свойства языков программирования, как понятность конструкций языка и простота языка.

Понятность конструкций языка — это свойство, обеспечивающее лёгкость восприятия программ человеком. Данное свойство языка программирования зависит от выбора ключевых слов и такой нотации, которая позволяла бы при чтении текста программы легко выделять основные понятия каждой конкретной части программы, а также от возможности построения модульных программ. Высокая степень понятности конструкций языка программирования позволяет быстрее находить ошибки. [5, стр. 10]

Простота языка обеспечивает лёгкость понимания языковых конструкций и запоминания синтаксиса языка. Простой язык предоставляет ясный, простой и единообразный набор понятий. При этом желательно иметь минимальное количество различных понятий с как можно более простыми и систематизированными правилами их комбинирования, т.е. язык должен обладать свойством концептуальной целостности [5, стр. 11].

Согласно рейтингу веб-ресурса TIOBE на март 2020 года, наибольшей популярностью среди разработчиков программного обеспечения пользуются следующие языки программирования:

1. Java
2. C
3. Python
4. C++
5. C# [6]

Данные языки программирования можно разделить на два класса: аппаратно-ориентированные (ориентированные на высокую производительность исполняемого кода) и ориентированные на высокую продуктивность и лёгкость работы, обладающие хорошей понимаемостью программ [7]. К первому классу можно отнести языки C, C++. Ко второму — Python (поэтому в данной статье в основном сравниваются возможности именно языков Python и C++). Есть также языки, занимающие промежуточное положение — Java, C#. Но успешных языков программирования, которые уверенно можно было бы отнести сразу к двум этим классам, фактически не существует.

Целью данной работы является исследование возможности улучшения языков программирования по критериям «понятность конструкций языка» и «простота языка» за счет использования когнитивно-эргономического подхода к проектированию синтаксиса языков программирования, в рамках чего предлагается структура нового языка программирования, сочетающего свойства языков обоих классов.

## 1. Отсутствие визуальных помех

В представленных в данном перечне языках программирования присутствуют визуальные помехи в виде обязательных символов «точка с запятой» в конце утверждений, фигурных скобок (в других языках — в виде ключевых слов BEGIN и END) для обозначения областей кода, либо символа «двоеточие».

Рассмотрим такой код на Python:

```
while True:
    if instr[i] == "[":
      nesting_level += 1
    elif instr[i] == "]":
      nesting_level -= 1
      if nesting_level == 0:
        break
    i += 1
```

Ему соответствует следующий код на C++:

```
while (true) {
    switch (instr[i]) {
    case '[':
       nesting_level++;
       break;
    case ']':
       if (--nesting_level == 0)
          goto break_;
       break;
    }
    i++;
    …
```

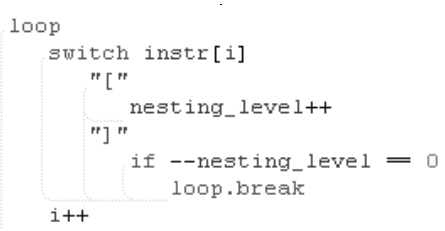



Рис. 1. Обозначение областей кода посредством пунктирных линий

Таблица 1. Ключевые слова Python:

| | | | | |
|---|---|---|---|---|
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

Таблица 2. Ключевые слова C++:

| | | | | | |
|---|---|---|---|---|---|
| alignas | char32_t | enum | namespace | return | try |
| alignof | class | explicit | new | short | typedef |
| and | compl | export | noexcept | signed | typeid |
| and_eq | const | extern | not | sizeof | typename |
| asm | constexpr | false | not_eq | static | union |
| auto | const_cast | float | nullptr | static_assert | unsigned |
| bitand | continue | for | operator | static_cast | using |
| bitor | decltype | friend | or | struct | virtual |
| bool | default | goto | or_eq | switch | void |
| break | delete | if | private | template | volatile |
| case | do | inline | protected | this | wchar_t |
| catch | double | int | public | thread_local | while |
| char | dynamic_cast | long | register | throw | xor |
| char16_t | else | mutable | reinterpret_cast | true | xor_eq |

```
}
break_:
```

Одной из ключевых черт нового предлагаемого в данной статье языка программирования является полное отсутствие визуальных помех. Ниже представлен соответствующий код на новом языке программирования:

```
loop
  switch instr[i]
    "["
      nesting_level++
    "]"
    if —nesting_level == 0
      loop.break
  i++
```

Для обозначения областей кода вместо языковых средств предлагается использовать на уровне среды разработки менее отвлекающие (нежели символы либо ключевые слова) пунктирные линии (рис. 1).

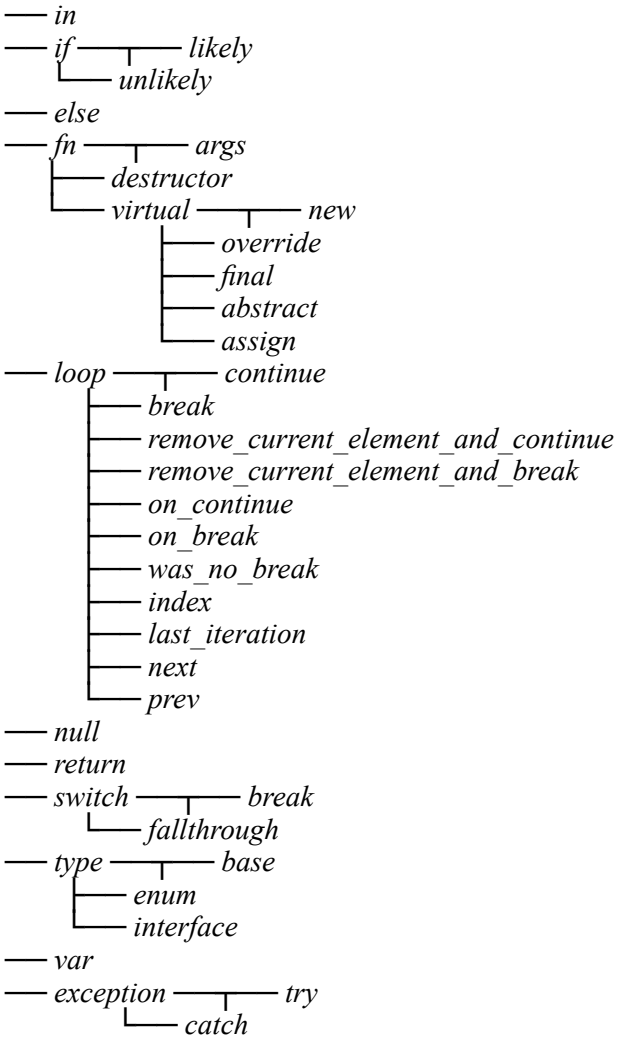
Мы видим, что данный код более компактен и не имеет отвлекающих знаков и символов.

## 2. Иерархическая организация ключевых слов языка

В языках программирования таких, как Python, C++, Java ключевые слова идут в виде списка (таблицы 1, 2).

Еще одна ключевая черта для нового предлагаемого языка программирования в том, что его зарезервированные слова идут не просто в виде списка, а структурированы в иерархию, на верхнем уровне которой располагаются 11 корневых/базовых ключевых/зарезервированных слов. При начальном изучении языков программирования это позволяет легче запомнить ключевые слова языка, связанные между собой.

Дерево всех ключевых слов нового языка программирования выглядит так:




## 3. Эргономичные обозначения операторов языка

В то время как у других языков программирования пояснение выбора того или иного оператора языка приводится крайне редко, все операторы нового языка программирования имеют рациональное обоснование их выбора. Рассмотрим некоторые операторы:

**Поразрядное исключающее ИЛИ**

В языках Python, C++, Java и многих других для обозначения данной операции используется символ ^.

В новом языке программирования для обозначения данной операции предлагается использовать тройку символов (+), так как они похожи на символ $\oplus$, который используется в математической логике для обозначения операции «исключающего или» [8].

И хотя $\oplus$ используется чаще для одноразрядных значений, в научных статьях можно встретить его применение для массивов из байт [9].

К тому же, использовать символ ^ для данной операции (как это сделано в Си) можно считать неудачной идеей, так как это сбивает с толку новичков в программировании [10]. Для символа ^ больше подходит операция возведения в степень.

**Логическое И**

В языке Python для данной операции используется ключевое слово and, а в языке C++ — символы &&.

Для данной операции в новом языке предлагается использовать символ & вместо общепринятого (то есть взятого из Си) && так как:

1. Логическое И требуется гораздо чаще, чем поразрядное.
2. Поразрядное И практически всегда используется только для проверки установленных флагов, но это чревато ошибками (так как в выражении flags & SOME_FLAG константа SOME_FLAG может вообще не иметь отношения к flags, и компилятор (языков Си, C++ и др.) пропустит такую ошибку), поэтому было принято такое решение, чтобы сподвигнуть отказаться от такого использования поразрядного И вообще.

**Поразрядное НЕ**

В языках Python и C++ для обозначения данной операции используется символ ~.

В предлагаемом новом языке программирования для его обозначения выбрана тройка символов (–), так как этот оператор, по сути, является парным оператору (+) и обладает следующими свойствами:

NOT a = 0(–)a = (–)a
a(–)(–)b = a(+)b
a(+)(–)b = a(–)b
(–)(–)a = a
a(–)b = NOT (a XOR b)

Однако, так как бинарный оператор (–) не имеет аппаратной поддержки и используется крайне редко, в новый язык включён только унарный оператор (–), который соответствует поразрядному НЕ.

**Возведение в степень**

Символ ^ для данной операции используется в языках BASIC, MATLAB, Lua, TeX (в свою очередь используется в <math> в Wikipedia), Julia.

Обозначение ** для данной операции используется в Perl, PHP, Python, Ruby, CoffeeScript. В языке D используется пара символов ^^. Но предположительно основной причиной отказа от ^ в качестве обозначения для оператора возведения в степень в этих языках является использование этого символа для операции поразрядного исключающего ИЛИ, что можно считать неудачной идеей [10].

Поэтому в новом языке для оператора возведения в степень был выбран символ ^.

**Целочисленное деление**

В языке Python для данной операции используется пара символов // [11].

В языке C++ отдельного оператора для целочисленного деления нет и для выполнения соответствующей операции необходимо использовать преобразование типа, например: int(x / y).

В новом языке для данного оператора предлагается пара символов I/ (I обозначает Integer — целочисленный), а пара символов // обозначает начало однострочного комментария, как в C++ [12].

**Конкатенация массивов**

В языке Python для данной операции используется символ +.

В языке C++ данная операция не имеет соответствующего символа и осуществляется посредством метода insert.

Для обозначения выбрана тройка символов [+], так как массивы в новом языке (также как в JavaScript, Python, Ruby, Swift и многих других языках) задаются с использованием квадратных скобок, например: [1, 2, …]. А так как эта операция достаточно ресурсоёмкая, ей выделен отдельный оператор (в других языках программирования используется оператор +).

Конкатенация строк. Оператор +, используемый во многих языках программирования (Python, C++, Ruby, Java, JavaScript), для данной операции можно считать неудачным решением по причине того, что он конфликтует с арифметическим оператором +, например:

System.out.println(«2 + 2 = « + 2+2);

Этот код на Java выведет 2 + 2 = 22 вместо ожидаемого 2 + 2 = 4.

Аналогично, возникнет проблема с записью «id=» + id+1 и т.п.

Предположительно по этой причине от + для конкатенации строк отказались во многих языках (PHP, Perl, D, Lua, Julia, Visual Basic, Nim, Dart), вот только однозначного обозначения для этой операции, к сожалению, до сих пор так и не определили (наиболее удачным можно считать принятое в языках D и Perl 6 обозначение — символ тильда ~), поэтому было принято решение пойти другим путём и вообще отказаться от оператора конкатенации строк. Для конкатенации строкового литерала и переменной в новом языке программирования достаточно просто расположить их рядом без пробелов:

print(“id=”id)
print(“id=” (id+1))

Для конкатенации двух переменных используется такая форма записи:

print(name»»value)

Для добавления к уже имеющейся строковой переменной используется оператор «»=:

res_str «»= «, «

При этом запись "2 + 2 = "2+2 в новом языке является некорректной, так как вначале вычисляется "2 + 2 = "2, что является строкой, а прибавление к строке числа

Таблица 3. Примеры использования ключевых подслов loop

| C++ | Новый язык программирования |
|---|---|
| for (auto el: {1, 3, 5})<br>printf("current element is%i\n", el); | loop(el) [1, 3, 5]<br>print("current element is "el) |
| int index = 0;<br>for (auto el: {1, 3, 5}) {<br>printf("%ith element is%i\n", index, el);<br>index++;<br>} | loop(el) [1, 3, 5]<br>print(loop.index"th element is "el) |
| bool was_found = false;<br>for (auto &&el: container)<br>if (condition(el)) {<br>printf("Element was found\n");<br>was_found = true;<br>break;<br>}<br>if (!was_found)<br>printf("Element was not found\n"); | loop(el) container<br>if condition(el)<br>print("Element was found")<br>loop.break<br>loop.was_no_break<br>print("Element was not found") |
| bool first_iteration = true;<br>for (auto el: {1, 3, 5}) {<br>if (!first_iteration)<br>printf(", ");<br>else<br>first_iteration = false;<br>printf(«%i», el);<br>} | loop(el) [1, 3, 5]<br>if!loop.first_iteration<br>print(", ", end'"")<br>print(el, end'"") |
| std:: vector<int> arr = {1, 3, 5};<br>for (auto it = arr.begin();<br>it!= arr.end();) {<br>if (*it == 3)<br>it = arr.erase(it);<br>else<br>++it;<br>} | var arr = [1, 3, 5]<br>loop(el) arr<br>if el == 3<br>loop.remove_current_element_<br>and_continue |

недопустимо. И в новом языке правильно писать так: "2 + 2 = "(2+2).

## 4. Циклы

В языке Python циклы реализуются посредством ключевых слов for и while. В языке C++ дополнительно к этим двум словам добавляется ещё do.

В то время как в функциональных языках программирования используются конструкции «for each» (запись times и each в Ruby и Groovy, а также foreach/forEach в Scala/Kotlin), использование отдельных ключевых слов предоставляет больше возможностей, чем такие конструкции. Например, возможность прервать цикл используя ключевое слово break. А потому в новом языке циклы предлагается выделить в целую группу ключевых подслов, объединённую корневым ключевым словом loop.

Предлагаемый новый язык программирования поддерживает 4 формы циклов:

**1. Бесконечный цикл.**

```
loop
    print("\a")
    sleep(1)
```

**2. Цикл «while».**

loop <условие>
…

Тело цикла выполняется/повторяется до тех пор, пока заданное условие истинно.

**3. Повтор тела цикла заданное количество раз.**

loop <число>
…
Пример:

```
loop 3
    print(«ABC»)
```

выведет «ABC» 3 раза.

**4. Цикл «for in».**

loop(<переменная>) <итерируемое выражение>

…

Примеры:

loop(x) 1.10 {print(x)} выведет числа от 1 до 10 включительно.

loop(c) "str" {print(c)} выведет "s", "t" и "r".

loop(n) [9,7] {print(n)} выведет 9 и 7.

**Ключевые подслова loop**

loop.break прерывает выполнение текущего цикла.

loop.continue прерывает выполнение текущей итерации текущего цикла, и продолжает его выполнение на следующей итерации.

То, что в других языках программирования требуется каждый раз реализовывать вручную {например, добавлять переменную-счётчик если требуется получить номер текущей итерации цикла обхода элементов контейнера (связного списка или map) или заводить дополнительные булевы переменные для определения того, был ли прерван цикл, первая ли сейчас итерация цикла и т.д., см. табл. 1}, в новом языке представлено в виде дополнительных ключевых подслов оператора loop:

loop.remove_current_element_and_continue удаляет текущий элемент итерируемого контейнера.

Код под loop.was_no_break выполняется только когда цикл успешно завершился без вызова loop.break.

loop.index можно использовать для получения индекса (номера начиная с 0) текущей итерации цикла.

loop.last_iteration является истиной, если текущий элемент итерируемого контейнера последний.

loop.first_iteration является истиной, если текущий элемент итерируемого контейнера первый.

loop.prev и loop.next обеспечивают доступ к предыдущему или следующему элементу итерируемого контейнера.

Форма записи loop(переменная) диапазон_или_объект была выбрана вместо loop(диапазон_или_объект) переменная или вместо loop переменная диапазон_или_объект для того, чтобы можно было писать loop(переменная).break. Например:

```
loop(y) 0.10
   loop(x) 0.10
     if a[y][x] == 1
        loop(y).break // прерывает внешний цикл по `y`
```

Прервать внешний цикл можно также с помощью записи ^loop.break, ^^loop.break и т.д. Такая запись более логична, чем break N в PHP [13], так как не очевидно с какого числа начинается отсчёт: с 0 или 1 (в PHP начинается с 1, т.е. просто break равнозначен break 1, а чтобы прервать внешний цикл, нужно использовать break 2).

## 5. «Синтаксическая соль»

Синтаксическая соль — это синтаксические возможности (конструкции) языка программирования, которые не добавляют какого-либо функционала, а предназначены для того, чтобы писать плохой код ({плохо читаемый и трудно сопровождаемый}) было труднее [14].

**Приведение типов**

Классическим примером общеизвестной и широко применяемой «синтаксической соли» являются имеющиеся почти в любом языке программирования со статической типизацией встроенные команды преобразования типов данных. Формально эти команды излишни (так как тип переменной-назначения итак известен компилятору), но в языках, где их применение обязательно, программист вынужден каждый раз обращать внимание на то, что он выполняет потенциально опасное смешение типов, которое часто указывает на логическую ошибку в программе.

Также как и другие языки со статической типизацией (C++, Java, Swift), в новом языке необходимо явное приведение типов. Например, чтобы присвоить переменной s строкового типа String значение целочисленной переменной i (другими словами — сконвертировать целочисленную переменную в строку), необходимо написать:

```
s = String(i)
```

**Входные и выходные параметры функций**

В языке Python параметры (аргументы) функций передаются по значению (и таким образом являются входными), за исключением объектов (включая списки), которые передаются по ссылке (и таким образом являются входными и выходными). В языке C++ для явного указания на то, что параметр функции может быть выходным (то есть, изменён внутри функции) используется синтаксис ссылок (символ & перед именем параметра функции), но при вызове самой функции это никак не указывается.

Также как в языке программирования Swift [15], в новом языке при вызовах функций для тех аргументов функций, которые могут быть изменены внутри них (так называемые in-out (входные и выходные) параметры), предлагается использовать префикс амперсанд (&):

```
tokenize(source, &comments)
```

При такой записи сразу видно, что функция tokenize во время своего выполнения может изменить переменную comments.

Таблица *4.* Добавление в массив числа на языках **Python** и **C++**

| Python | C++ |
|---|---|
| array1 = []<br>array2 = array1<br>array2.append(1)<br>print(len(array1)) # выведет 1 | vector<int> array1;<br>vector<int> array2 = array1;<br>array2.push_back(1);<br>cout << array1.size(); // выведет 0 |

**Префиксы имён переменных**

В некоторых языках программирования для выделения области переменных иногда используются алфавитные префиксы. Пример — венгерская нотация в C++ предписывает всем глобальным переменным к имени добавлять префикс g_, а к переменным членам классов — префикс m_. Однако, такая форма записи недостаточно изящна, а потому не нашла широкого применения. В предлагаемом языке программирования используется более компактная и наглядная запись: глобальные переменные необходимо выделять префиксом: (по аналогии с:: в C++ — например,:: global_var), а переменные-члены — префиксом. (в предлагаемом языке указатель this/self обозначается тройкой символов (.), и таким образом (.).member_var можно сократить до.member_var, в то время как в некоторых других языках (например, Python и Rust) необходимо писать self. member_var).

Символ @ в новом языке программирования используется для внешних (**ca**ptured) переменных, так как он похож внешне на английские буквы C и a. В языке Python для этого используется ключевое слово nonlocal, а в C++ — список захвата (capture list).

Назначение префиксов переменных — такая же, как и у венгерской нотации — повышение читаемости кода.

**Спецификаторы виртуальных функций**

В языке Python спецификаторы виртуальных функций не предусмотрены, так как у него динамическая типизация и приближённо можно сказать, что все функции (методы) в Python — виртуальные. В языке C++ обязательно указывать, является ли функция виртуальной. Также предлагается делать и в новом языке программирования. Но, помимо этого, необходимо указать «тип её виртуальности». Возможно 5 вариантов:

1. new — это новая виртуальная функция. Наличие виртуальной функции с этим же именем в базовом классе является ошибкой компиляции.
2. override — это переопределяющая виртуальная функция. Отсутствие виртуальной функции с этим же именем в базовом классе является ошибкой компиляции.
3. final — это финальная виртуальная функция. Переопределение этой функции в производных классах запрещено.
4. abstract — это абстрактная виртуальная функция. Её необходимо определить в производном классе. Инстанцирование объектов класса, содержащего хотя бы одну абстрактную виртуальную функцию, запрещено (другими словами, невозможно создавать объекты таких классов — такие классы можно использовать только как основу для производных классов).
5. assign — это определение/реализация абстрактной функции.

В C++ представлены все эти варианты [16], кроме new и assign, однако спецификатор override является необязательным в C++ и обязательным в новом языке.

**Оператор присваивания для сложных типов**

Существующие языки программирования чётко определяют поведение оператора присваивания, обозначаемого символом равно (=). Но, в то время как в одних языках программирования (например, C++) этот оператор обозначает поэлементное копирование объектов, в других языках (Java, Python, Ruby), происходит лишь копирование ссылки на объект.

Рассмотрим это на примере (см. табл. 4).

Как можно заметить, поведение Python несколько сбивает с толку (так как добавление в массив array2 изменяет также и array1). А поведение в C++ скрывает такую ресурсоёмкую операцию, как поэлементное копирова-

ние массивов, за простые знакомы равенства. Поэтому в новом предлагаемом языке программирования операция присваивания для сложных типов не определена, и необходимо в каждом конкретном случае явно указывать требуемое поведение — копирование либо создание новой ссылки. Так, чтобы получить поведение как в C++ (копирование), необходимо вместо array2 = array1 написать array2 = copy(array1).

Чтобы получить поведение как в Python напишем: array2 = share(array1).

## Заключение

В данной статье были рассмотрены основные черты нового разрабатываемого языка, удовлетворяющие требованиям когнитивной эргономики. Были рассмотрены такие элементы синтаксиса нового языка программирования как: отсутствие визуальных помех, иерархическая организация ключевых слов языка, эргономичные обозначения операторов, циклы, приведение типов, префиксы имён переменных, спецификаторы виртуальных функций.

Для обозначения логических операторов поразрядное исключающее ИЛИ, а также возведение в степень были выбраны соответствующие обозначения (+) и ^, являющиеся эргономичными исходя из практики использования соответствующих математических обозначений.

Операции конкатенации массивов и строк не вызывают конфликта с операцией + и также интуитивно понятны для восприятия.

Все вышеперечисленные особенности позволяют упростить язык для восприятия программистом по критериям «понятность конструкций языка» и «простота языка», сделать его интуитивно понятным, что в свою очередь позволит более эффективно решать поставленные задачи.

ЛИТЕРАТУРА


1. Пароджанов В. Д. Как улучшить работу ума: Алгоритмы без программистов — это очень просто! Москва: Дело, 2001. 360 с.
2. Schröder E. Vorlesungen über die Algebra der Logik. Bd I. Leipzig, 1890.
3. Марр Д. Зрение. Информационный подход к изучению представления и обработки зрительных образов: Пер. с англ. Москва: Радио и связь, 1987. 400 с.
4. Пароджанов В. Д. Как написать хороший учебник для хороших людей. Учебники, о которых мечтают студенты и школьники. Москва: ДМК Пресс, 2017. 500 с.
5. Молдованова, О. В. Языки программирования и методы трансляции: учебное пособие / О. В. Молдованова. — Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2012. — 134 с. — URL: http://www.iprbookshop.ru/54809.html
6. TIOBE Index for March 2020 URL: https://www.tiobe.com/tiobe-index/
7. Белова И. Язык программирования Python URL: https://youtu.be/6I7ybevPUKM?t=42
8. Зыков А.Г., Поляков В. И., Скорубский В. И. Математическая логика. — СПб: НИУ ИТМО, 2013. 131 с.
9. Bellare M., Canetti R., Krawczyk H. Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996).
10. What is ^ used for in ruby? URL: https://stackoverflow.com/questions/11464905/what-is-used-for-in-ruby
11. Expressions — Python 3.7.4 documentation URL: https://docs.python.org/3/reference/expressions.html#binary-arithmetic-operations
12. Comments — cppreference.com URL: https://en.cppreference.com/w/cpp/comment
13. PHP: break — Manual URL: https://www.php.net/manual/en/control-structures.break.php
14. Syntactic salt URL: http://www.catb.org/jargon/html/S/syntactic-salt.html
15. Functions — The Swift Programming Language (Swift 5.1) URL: https://docs.swift.org/swift-book/LanguageGuide/Functions.html#ID173
16. Virtual function specifier — cppreference.com URL: https://en.cppreference.com/w/cpp/language/virtual