Годишник на Технически Университет - София, том 69, книга 2, 2019
Proceedings of the Technical University of Sofia, Volume 69, Issue 2, 2019

ЕЗИЦИ ЗА ПРОГРАМИРАНЕ НА ИЗКУСТВЕН ИНТЕЛЕКТ

Данаил Славов

Резюме: *Настоящата работа представя обзор на някои от най-използваните езици за програмиране в областта на изкуствения интелект. Разглеждат се техни предимства и недостатъци, както и поддържаните от тях парадигми на програмиране. Отчетени са съображенията при избор на език в зависимост от конкретната желана функционалност на разработваното решение. Описана е реализацията на някои от популярните подходи и методи за програмиране на системи с изкуствен интелект и са обобщени най-подходящите приложения на възможностите им.*
Ключови думи: *изкуствен интелект, езици за програмиране, машинно обучение, парадигми на програмиране, физическа знакова система*

PROGRAMMING LANGUAGES FOR ARTIFICIAL INTELLIGENCE

Danail Slavov

Abstract: *This paper provides an overview of some of the most commonly used programming languages in the field of artificial intelligence (AI), as well as their capabilities and limitations. The programming paradigms supported by these languages are examined. A set of considerations for choosing a particular language is given, depending on the specific desired functionality of the solution being developed. The work also includes a description of the realization of popular programming strategies and approaches used in AI system development. The most suitable applications of each language potentialities are summarized.*
Keywords: *artificial intelligence, programming languages, programming paradigms, physical symbol system*

1. ВЪВЕДЕНИЕ

Изкуственият интелект (ИИ) основно е насочен към разработване на методи за реализиране на аспекти от човешкото интелигентно поведение. Той се стреми не само да изучава и да разбира интелекта, но най-вече да синтезира и изгражда полезни за практиката системи, притежаващи такова интелигентно поведение [2]. Този дял от компютърните науки извиква конкретни нужди в сферата на езиците за програмиране. Важна част на езиците за ИИ е, че предоставят възможност за имплементиране на физически знакови (символни) системи (ФЗС), тъй като те притежават необходимите и достатъчни средства за осъществяване на интелигентно поведение. Тази „централна работна догма“ в ИИ предполага, че мисловните процеси на човека са аналогични на някакъв вид знакови изчис-

ISSN 1311-0829

лителни процеси, разбирани като преобразуване на синтактично правилни знакови структури чрез прилагане на съответни трансформационни правила в рамките на някаква ФЗС[2].
Също като при разработването на повечето софтуерни приложения, програмистите разполагат с различни езици, които да използват при създаването на ИИ. Не може обаче да се открои само един от тях като съвършения програмен език, използван в изкуствения интелект. Процесът на разработване зависи от желаната функционалност на замисленото крайно приложение. Ето защо настоящата работа представя четири от най-използваните в тази сфера езици заедно с техни силни страни и някои характеристики, приемани като недостатъци.
В идеалния случай програмният език за ИИ трябва да осигурява механизми за представяне и манипулиране на познание от реалния свят. Това обикновено се осъществява чрез използване на логически формализъм, позволяващ достигане до заключения[2].

2. LISP

Възникнал още през 1958 г., LISP е вторият (след Fortran, 1957 г.) най-стар език за програмиране, който се използва и до днес. Първоначално създаден като практическа математическа нотация за компютърни програми, повлиян от ламбда-смятането на Алонсо Чърч [Alonzo Church], бързо става предпочитан програмен език за изследвания в областта на ИИ. LISP проправя път за много идеи в компютърните науки, включително реализиране на дървовидни структури от данни, автоматично управление на хранилищата, динамично типизиране, условни конструкции, реализиране на рекурсия и цикъл „четене-оценяване-извод“ (read-eval-print)[9].
В LISP има само две нива обекти на данни. Атомите са данни в най-проста форма и представляват низ от знаци, използван за символно представяне. Структурата от по-високото ниво се нарича S-израз (symbolic expression, символен израз). Един S-израз може да бъде или единичен атом, или свързан списък от S-изрази, и следователно е способен да кодира в себе си данни с произволна дървовидна структура:
(ivan, petar, mihaela)
(students (ivan, petar, mihaela)).
Всички програми и всички данни в LISP спазват структурата на S-изрази и могат да се представят като вложени списъци. Макар традиционно да се смята, че за да бъде един език по-полезен, той трябва да предоставя разнообразни структури от данни, философията на LISP е да предоставя една-единствена мощна структура и да позволява на потребителя да се фокусира върху задачата, без да се обременява с поддържането на разнообразни синтактични и типови наредби.
В много случаи първият елемент от списъка се възприема като име на функция, а останалите – като нейни аргументи. Съществуват функции, които позволяват дефиниране на нови функции. Новодефинираните функции се представят с помощта на λ-смятането на Чърч. Примерът по-долу показва дефиниране на функция за изчисляване на площта на кръг:

```
(def circle-area
   (lambda (r)
      (times pi (times r r)))).
```

Функцията „def“ не оценява аргументите си, а само свързва втория елемент на списъка (в този случай атома „circle-area“) с λ-израза, представляващ третия елемент.

При реализацията на база данни в LISP всеки атом може да е свързан със списък от двойки свойство-стойност, познат като „p-списък“ (property list, списък от свойства). Една от честите употреби на p-списъците в програмирането на ИИ е като средство за реализиране на мрежи – атомите представят възлите, а свойствата са означените ребра. Необходимостта от по-всеобхватни възможности за работа с базите данни мотивира подхода за обособяване на знанието за дадена предметна област в съвкупност от твърдения (или наредени n-орки), всяко от което представлява факт, вместо да се натрупват факти като свойства[1]. Например:

```
(student name ivan)
(teacher class robotics).
```

Един от полезните методи за достъп до база данни е нотацията на съвпадение по шаблон (pattern matching). В най-простия си вариант този метод съставя шаблон, който е структурно подобен на дадено твърдение, но може да включва променливи, които трябва съгласувано да се заменят от атоми, присъстващи в твърдение от базата данни, така че да се получи съвпадение. Например следните шаблон и твърдение съвпадат след субституция на променливата *x* с атома *ivan*:

```
(student name ?x)
(student name ivan).
```

Необходимостта от поява на още по-силно формализирани възможности за програмиране, включващи търсене с възврат при използване на база данни от твърдения, налага появата на система за програмиране, основана на формалната логика[1].

3. Prolog

Идеята за изграждане на компютърно-изпълними програми, чийто инструментариум включва приложението на предикатното смятане и принципа на резолюцията (т.е. програми, които изпълняват логически формули), се оказва толкова плодотворна и мощна, че става парадигма на подхода към логическо програмиране. Най-разпространеният в момента език, който поддържа тази парадигма, е Prolog[2] (programmation en logique (фр.), programming in logic (англ.), програмиране в логиката), създаден около 1972 г. от Ален Колмерое [Alain Colmerauer] и Филип Ръсел [Philippe Roussel] въз основа на процедурната интерпретация на Робърт Ковалски [Robert Kowalski] относно клаузите на Хорн[10].

От синтактична гледна точка всяка програма в Prolog представлява множество от обратни клаузи на Хорн, т.е. от вида u ← p ∧ q ∧ r ∧ ... ∧ s. За удобство знакът за импликация се заменя с двоеточие и тире, а знакът за конюнкция – със запетая: u :- p, q, r, ..., s. Разграничават се четири вида клаузи:

- факти: единични литерали от вида u :-

- правила: положителни клаузи от вида u :- p, q, r, ..., s
- цели (въпроси, заявки): отрицателни клаузи от вида :- p, q, r, ..., s
- празна клауза (без литерали) от вида :-

Фактите са формули, които се приемат за безусловно истинни в дадена предметна област. Правилата са формули, които са истинни, при условие че конюнкцията в тялото (дясната част) на правилото е истина. Отрицателните клаузи представляват инвертирани цели, а празната клауза означава противоречие.

При конструирането на една програма всеки програмист има в съзнанието си някаква възнамерявана интерпретация (предметна област), с елементите на която свързва избраните от него имена на константи, функции и предикати. Стремежът му е да създаде база данни, към която да се отправят въпроси и решенията да се получават като нейни логически следствия[2]. Например от следната база:

likes(adam, X) :- likes(X, apples).
likes(eve, apples).

отговорът на въпроса „Какво харесва Адам?“, т.е. :- likes(adam, X), е логическото следствие likes(adam, eve).

Декларативният смисъл на една логическа програма е множеството от всички основни факти, които са нейни логически следствия. Освен декларативно обратните клаузи на Хорн могат да се разглеждат и процедурно: главата u на клаузата u :- p, q, r, ..., s се третира като име на процедура, изпълнението на която изисква изпълнение на процедурите p, q, r, ..., s, специфицирани от литералите в нейното тяло. Тук обаче, поради изискването за наредба на литералите, комутативността вече не е в сила. Аргументите на главата u (ако има такива) служат като входно-изходни параметри на процедурата, т.е. чрез тях се предават/приемат стойности към/от p, q, r, ..., s, които се разглеждат като оператори за извикване на съответни процедури. Всеки литерал от отрицателната клауза (целта) се интерпретира като оператор за първоначално извикване на съответната му процедура и се нарича подцел.

При такава постановка една база данни, състояща се от обратни клаузи на Хорн, може да се интерпретира от гледна точка на процеса на логическо извеждане не само в декларативен, но и в процедурен смисъл, т.е. като логическа програма, която изпълнява логическите формули от базата.

Обобщено погледнато, всяка програма в Prolog и нейното изпълнение може да се представят схематично по следния начин:

ПРОГРАМА В PROLOG = ПРАВИЛА + ФАКТИ
ИЗПЪЛНЕНИЕ = ЛОГИЧЕСКО ИЗВЕЖДАНЕ [2]

Едно от различията между Prolog, като език за логическо програмиране, и останалите програмни езици, поддържащи други парадигми, е, че момента на извикване (Call) не е известно коя входна точка на процедурата ще бъде използвана, т.е. съществува недетерминизъм. Освен това е възможно повторно влизане във вече извикана процедура. Недетерминизмът се проявява при неуспешно завършване на процедурата (Fail) – задейства се възвратен механизъм, който автоматично реактивира (Redo) предходно извикана и вече изпълнена процедура:

Prolog основно се използва при решаване на логически задачи, а не при числови операции. Едно от най-успешните му приложения е за изграждане на експертни системи, които решават сложни проблеми без човешка намеса – напр. автоматично планиране, наблюдение, управление и отстраняване на неизправности в комплексни системи. С този език могат да се решават конкретни задачи, които се възползват от основаните на правила логически заявки: търсене в бази данни, гласов потребителски интерфейс, попълване на шаблони. Първоначалното му предназначение е за обработка на естествен език и често се използва при съвпадение по шаблон върху дърва на разбор (parse trees) за естествен език, тъй като удобно изразява тези правила на съвпадение и предоставя технология за ефективното им изпълняване.

4. C++

C++ е програмен език с общо предназначение от високо ниво, разработен през 1983 г. от Бярне Строуструп [Bjarne Stroustrup] като разширение на езика C. Той поддържа императивно, обектно-ориентирано и генерично програмиране, като също така предоставя възможности за работа с паметта на ниско ниво. Почти винаги се реализира като компилируем език и много доставчици предлагат компилатори за C++, така че може да се използва на различни платформи[11].
Основата на езика C (работещ на много по-ниско ниво), върху която е изграден C++, предоставя редица полезни възможности за подобряване на бързината, които в езиците от високо ниво обикновено липсват:

- При отсъствие на виртуални функции за структурите в C++ не се заделя допълнителна памет, както например за обектите в Java.
- Съществува възможност за създаване на временни масиви/обекти в стека (stack). Това може да се извърши без никакъв излишък на памет. Езиците от високо ниво обикновено принуждават системата да заделя пространство в хийпа (heap), чието управление е много по-сложно и следователно работата с него е по-бавна.
- C++ може да използва масиви от обекти – те се различават от масивите с указатели към обекти, каквито обикновено се използват в другите езици от високо ниво. Това намалява нуждата от указатели, спестява памет и води до едно ниво по-малко индиректност (дереферииране).
- Масивите с локален обхват (т.е. декларираните в рамките на функция) по подразбиране остават неинициализирани – нито един от елементите им не получава конкретна стойност. С други думи съдържанието на един масив е

неопределено в момента на декларирането му. Това може да спести много време в сравнение с подхода, при който всички стойности първоначално получават стойност нула (както е при Java).

- В много езици се извършва проверка на границите на променливите преди употребата им с цел да се гарантира например, че дадено число принадлежи към определен тип (проверка на обхвата) или че променлива, използвана за индекс на масив, е в границите на масива (проверка на индекса). Липсата на такива проверки в C++ допълнително ускорява изпълнението.

Очевидно е, че повечето от гореописаните свойства дават свобода на програмиста за взаимодействие със системата (най-вече с паметта), което способства за постигане на бързина, но също така и за по-лесно допускане на програмни грешки – съществува мнение, че работата със C++ не е така „безопасна“, както с други езици.

Тъй като в програмирането на ИИ широко се използват алгоритми, възможностите за реализирането им чрез програмния език имат съществено значение. C++ използва едни от най-удобните и полезни за ИИ структури от данни, а много от популярните алгоритми се съдържат в готов вид в стандартната библиотека STL. В този език програмистът има възможност да управлява изпълнението на ниско ниво чрез указатели, което е изключително удобно при програмирането на алгоритми.

Досега изброените преимущества на C++ го правят предпочитан език за приложения с невронни мрежи и машинно обучение, където се налага усилено използване на статистически техники и обработване на големи количества данни[6].

В много случаи програмите с ИИ са предназначени за изпълнение от вградени системи и устройства, например мобилни роботи и камери за техническо зрение, реализирани с микроконтролери, при които изборът на програмен език често пъти е ограничен до C и C++.

Сред недостатъците на езика C++ може да се отбележи нуждата от повече усилия за изграждане на фундаменталната част на програмата (т.нар. „boilerplate“), преди да се пристъпи към по-конкретните реализации. Също така, поради многостранните и задълбочени възможности, които предлага, този език е по-труден за научаване и използване от по-неопитните програмисти.

Бързината на C++ се използва широко в проекти за програмиране на ИИ, при които времето за изпълнение е от критично значение, като търсещи машини и компютърни игри. Много от библиотеките за други езици (включително Python) са написани на C++ или C именно поради внушителното им бързодействие и производителност.

5. Python

Езикът Python е създаден от нидерландския програмист Гуидо ван Росум [Guido Van Rossum] през 1991 г. Python е многопарадигмен език: напълно поддържа обектно-ориентирано и структурно програмиране, а много от функциите му поддържат функционално и аспектно-ориентирано програмиране (включително метапрограмиране и метаобекти). Вместо цялата функционалност да е вградена в яд-

рото му, Python има големи възможности за разширение (някои от които поддържат контрактно и логическо програмиране). Компактната му модулна архитектура го прави особено популярен като средство за добавяне на програмируеми интерфейси към съществуващи приложения[12].

Езикът Python се интерпретира, което означава, че не е необходимо да бъде компилиран към инструкции на машинен език преди изпълнението си и може да се използва директно за изпълнение на програмата. Това е достатъчно, за да бъде интерпретиран от емулатор или виртуална машина, базирана на машинен код, който хардуерът разбира. Интерпретируемите езици обикновено са по-бързи за прототипизиране на кода, но в определени случаи не се възползват от някои оптимизации и предимства в ефективността при изпълнение на компилиран код[4].

Python е динамично типизиран – всяко име на променлива е свързано само с даден обект (който на свой ред е от определен тип). Имената се свързват с обекти по време на изпълнение посредством присвояване, но при изпълнението на програмата е възможно дадено име да се свърже последователно с обекти от различни типове. При статично типизираните езици, като Java, C и C++, имената на променливите освен с обект са свързани и с определен тип (по време на компилиране, посредством деклариране на данните), като типът на променливата и типът на нейната стойност (обекта, с който е свързана) задължително трябва да съвпадат. Освен това Python е силно типизиран, тъй като интерпретаторът следи всички типове променливи. Това означава, че не може да се извършват операции, които са неподходящи за типа на обекта и биха имали непредсказуем резултат - например добавяне на число към низ - и това до някаква степен предпазва програмиста от внасяне на програмни грешки (бъгове) [8].

Все по-усиленото включване на графичния процесор в изчисленията, предлагащо възможности за паралелизъм между него и централния процесор, води до създаване на библиотеки като CUDA Python и cuDNN, които ускоряват обработването на данни от програми, написани на Python. Това улеснява изпълнението на операции, които изискват големи процесорни ресурси, като дълбоко обучение (deep learning), аналитични и инженерни приложения, и по този начин доближава по бързина Python до езици като C++[3].

Необходимостта от разработване на ефективни методи, които оценяват последствията от избора на дизайн при създаването на системи с ИИ, може да се разглежда като метазадача „взаимодействие-дизайн“, т.е. не само разработване на изкуствен интелект, но и проектиране на начините, по които той да бъде разработван[5]. В разрешаването ѝ може да помогне т.нар. софтуерно прототипизиране – дейност по създаване на прототипи на софтуерни приложения или непълни версии на разработваната софтуерна програма с цел да бъде изследвана преди окончателното ѝ завършване. Прототипът обикновено симулира само някои аспекти от крайния продукт и може да е напълно различен от него. Езикът Python предлага редица улеснения в хода на този процес – динамичното типизиране позволява на програмиста да съставя функции, които са достатъчно генерични за работа с всеки тип данни. Например:

```
def max_val(a,b):
    return a if a > b else b
```

Тази функция може да приема цели или дробни числа, низове, списъци, речници и т.н. Не е необходимо колекциите от данни в Python да са хомогенни, т.е. стойностите на всички елементи да са от еднакъв тип. Това позволява пакетиране на данните по уникален начин в движение и по-нататъшното им трансформиране в клас или структура в статично типизиран език като C++. Алгоритмите за работа със структури от данни са интуитивни и лесни за употреба, като също така правят кода по-четим. Например създаването на списък и проверката за конкретен елемент в него би изглеждало по следния начин:

```
list = [1,2,3]        # създава списък list с елементи 1, 2 и 3
result = 2 in list    # проверява дали в списъка list има елемент със стойност 2
```

което в C++ може да има следния вид:

```
list lst;
lst.push_back(3);
lst.push_back(1);
lst.push_back(7);
list::iterator result = find(lst.begin(), lst.end(), 7);
bool res = (result != lst.end());
```

Реализирането на ИИ изисква усилено използване на алгоритми. Описанието на дадена логика или функционалност може да се извърши в Python с до 1/5 по-малко код, отколкото при някои други езици, широко използвани в програмирането на ИИ. Интерпретационният му подход позволява извършване на проверки още по време на програмирането на алгоритмите.

Сред останалите предимства може да се открои фактът, че Python е с напълно отворен код и има огромна общност от потребители, което способства за наличието на разнообразни библиотеки, разширяващи функционалността на езика, включително и в сферата на ИИ.

За използването на Python в проекти с изкуствен интелект спомагат библиотеки като TensorFlow (за програмиране въз основа на потоци от данни и диференцирано програмиране) Keras (за невронни мрежи) и Scikit-learn (за машинно обучение). Като скриптов език с модулна архитектура, лесен синтаксис и инструменти за обработване на обогатен текст Python често се използва при обработване на естествен език.

Най-подходящите приложения на Python са в областите на машинното (само)обучение и дълбокото обучение и свързаните с тях реализации – например предсказващи системи при превозване на хора и товари, управление на рисковете във финансовата сфера, превенция и диагностициране на заболявания.

6. СЪОБРАЖЕНИЯ ПРИ ИЗБОР НА ПОДХОДЯЩИЯ ЕЗИК ЗА ПРОГРАМИРАНЕ

Когато става въпрос за реализиране на ИИ, общността от специалисти в тази област е разделена в мненията си кой език е най-подходящ за създаване на програми и приложения. Често програмистът избира език съобразно това, доколко комфортно се чувства при използването му, познанията и предишния си опит. Макар това да помага за по-бързо прототипизиране и разработване, в дългосрочен план може да не е най-добрият избор при изграждане на модели въз основа на прак-

тическия проблем, който трябва да се реши, и количеството на данните, които трябва да се обработват[7].
Езиците, които се компилират, най-често осигуряват по-добра цялостна производителност в сравнение с тези, които се интерпретират. Това се дължи на факта, че при компилирането кодът на програмата се редуцира до набор от машинни инструкции, след което се запазва като изпълним файл. При интерпретирането кодът запазва формата си и впоследствие се редуцира до инструкции едва по време на изпълнението. Интерпретируемите езици обаче предоставят допълнителни възможности - например програмите могат да се самоизменят чрез добавяне или променяне на функции по време на изпълнение. Също така процесът на разработване обикновено е по-лесен и бърз в интерпретируема среда, тъй като не нужно приложението да бъде прекомпилирано за всяко тестване дори на малка функционалност.
Лекотата, с която се използва даден език, зависи от много фактори, един от които са ограниченията за типа на данните. Динамичното типизиране позволява по-бързо разработване на приложенията, но при езиците със статични типове на данните се постига по-лесно преработване на нефункционални елементи от кода (рефакториране) и поддръжка на приложенията в дългосрочен план. Друг фактор е степента на многословие (verbosity), необходима за описание на дадена логика.
Важно съображение също така е състоянието на общността от програмисти, допринасящи за развитието на даден език, в която например може да се споделят идеи за бъдещи подобрения и да се търсят решения на различни проблеми.
Изборът на език за ИИ в крайна сметка зависи от конкретните задачи, които трябва да се изпълнят, сложността на реализацията, количеството обработвани данни, мащаба на планираното решение, предишния опит и възможностите на програмиста.

7. ЗАКЛЮЧЕНИЕ

Изкуственият интелект е интердисциплинарна област, която предлага необятни възможности за изследване и практически приложения. Наличното разнообразие от програмни езици и усъвършенстването на характеристиките им правят труден избора на език за програмиране на интелектни системи.
От настоящия обзор може да се заключи, че най-подходящият от представените езици за решаване на строго логически проблеми е Prolog, чиято структура и синтаксис наподобяват в най-силна степен естествената логика. Той обаче не се справя удовлетворително с изчислителни операции над числови обекти. В това отношение голямо преимущество има C++, който освен бързината си предлага и възможности за управление на изпълнението от по-ниско ниво. Езикът Python непрестанно печели поддръжници, привлечени от лесното му изучаване и употреба, което от своя страна съкращава времето, необходимо за постигане на крайните цели. Той също така може да изпълнява разнообразни функции достатъчно добре и бързо благодарение на множеството налични библиотеки.
В създаването на една цялостна програма, реализираща изкуствен интелект, нерядко се използват няколко програмни езика. Стремежът е да се постигне оптимално оползотворяване на възможностите, предлагани от всички тях, като съ-

щевременно се избегнат по-значимите им недостатъци. Честа практика особено при по-големи проекти е оформящата структура да се реализира с подходящо избран, удобен или просто предпочитан език от високо ниво, към която чрез интерфейси се свързват отделни модули, написани на различни езици в зависимост от конкретната функция.

ЛИТЕРАТУРА

[1] J.Glasgow, R. Browse, Programming Languages For Artificial Intelligence, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, Computers & Mathematics with Applications Vol. 11, No. 5, pp. 431–448, 1985

[2] Д. Димитров, Д. Никовски, Изкуствен интелект, второ преработено издание, Издателски комплекс на Технически Университет-София, 1999

[3] Rucha Urdhwareshe, Role of Python in Artificial Intelligence (AI), Cuelogic, June 23, 2016 – *https://www.cuelogic.com/blog/role-of-python-in-artificial-intelligence*

[4] Alex Castrounis, Python vs R for Artificial Intelligence, Machine Learning, and Data Science, *https://www.innoarchitech.com/python-vs-or-r-artificial-intelligence-ai-machine-learning-data-science-which-use/*, достъпен на 21.03.2019 г.

[5] Philip van Allen, Prototyping Ways of Prototyping AI, ACM Interactions, XXV.6 November-December, 2018, *http://interactions.acm.org/archive/view/november-december-2018/prototyping-ways-of-prototyping-ai*

[6] AI Programming: 5 Most Popular AI Programming Languages, February 7, 2018 – *https://existek.com/blog/ai-programming-and-ai-programming-languages/*

[7] Sandeep Vaid, Choosing the right programming language for machine learning algorithms with Apache Spark, June 29, 2018,*https://blogs.opentext.com/choosing-the-right-programming-language-for-machine-learning-algorithms-with-apache-spark/*

[8] *https://wiki.python.org*, достъпен на 21.03.2019 г.

[9] Lisp (programming language) *https://en.wikipedia.org/wiki/Lisp_(programming_language)*, достъпен на 20.03.2019 г.

[10] Prolog, *https://en.wikipedia.org/wiki/Prolog*, достъпен на 20.03.2019 г.

[11] C++, *https://en.wikipedia.org/wiki/C%2B%2B*, достъпен на 19.03.2019 г.

[12] Python (programming language) *https://en.wikipedia.org/wiki/Python_(programming_language)*, достъпен на 17.03.2019

Автор: Данаил Славов, ас. инж., катедра Автоматизация на електрозадвижванията; Факултет Автоматика, Технически Университет-София; E-mail address: *d.slavov@tu-sofia.bg*

Постъпила на 25.03.2019 г. **Рецензент**: доц. д-р Пенчо Венков