Практичне завдання 1

Курс:
Розробка вебсторінок мовою розмітки HTML5
з використанням каскадних таблиць стилів CSS3

Модуль 3. Графіка у web-дизайні. Оптимізація графіки. Гіперпосилання. Принципи навігації web-сайту

Завдання 1

Реалізуйте HTML-сторінку з текстом і фоном.

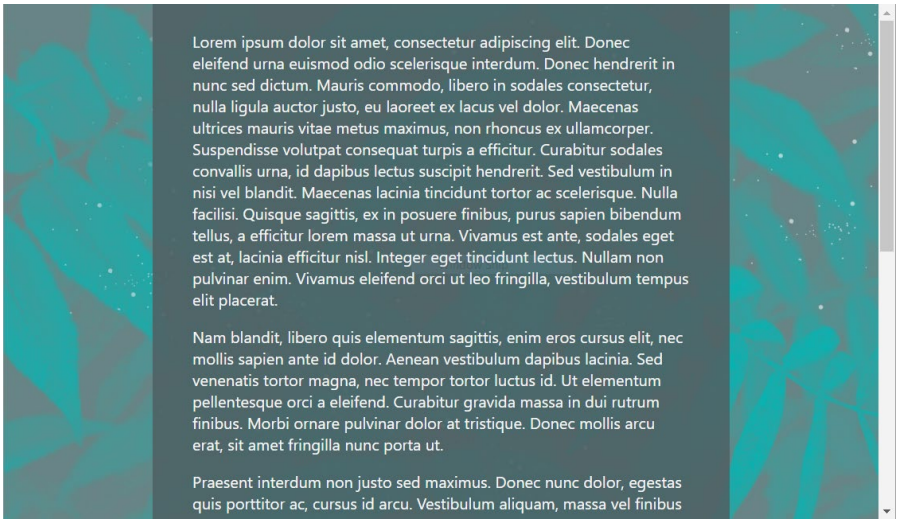
Вимоги:

- кількість тексту має бути достатньою для появи вертикального скрола на сторінці;
- фон з малюнком не повинен рухатися при скролі;
- однотонний фон під текстом має бути трохи прозорим.

Фонове зображення оберіть самостійно, а текст згенеруйте за допомогою цього сайту https://www.lipsum.com/.

* Щоб отримати доступ до матеріалів, відкрийте завдання в програмі Adobe Acrobat Reader.

Приклад кінцевого результату:



Завдання 2

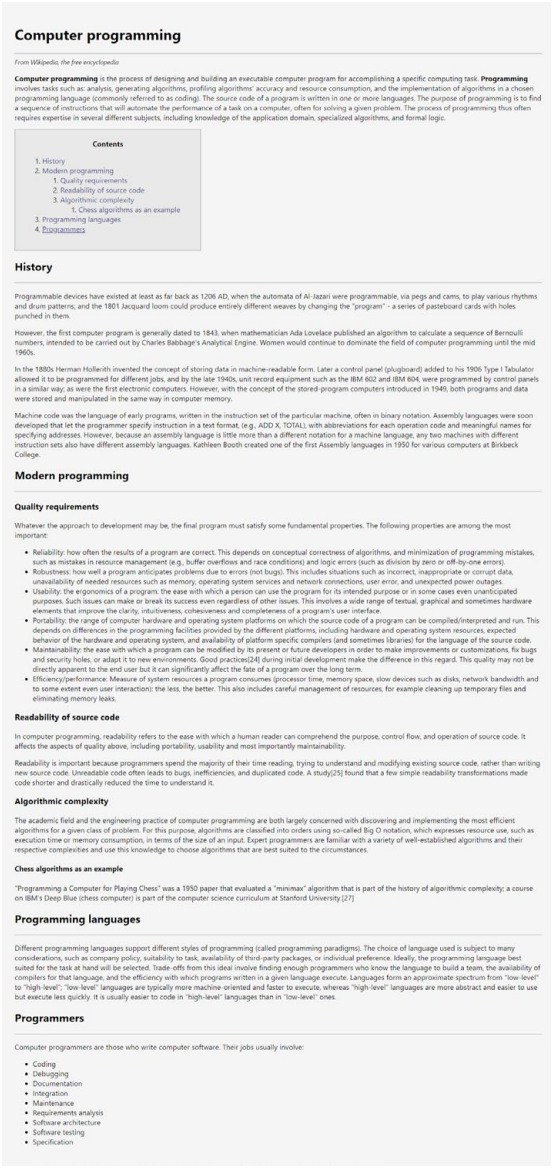
Реалізуйте HTML-сторінку про програмування з Вікіпедії.

Вимоги:

- при натисканні на посилання з розділу **Contents**, сторінка повинна прокручуватися до відповідного розділу;
- при наведенні мишкою на посилання з розділу **Contents**, напис посилання має підкреслюватися (як посилання **Programmers** на скриншоті на сторінці 3);
- стилізувати текст необхідно, як на скриншоті на сторінці 3.

Текст для виконання завдання прикріплений до даного pdf-файлу.*

Приклад кінцевого результату:



Computer programming

From Wikipedia, the free encyclopedia

Computer programming is the process of designing and building an executable computer program for accomplishing a specific computing task. **Programming** involves tasks such as: analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms in a chosen programming language (commonly referred to as coding). The source code of a program is written in one or more languages. The purpose of programming is to find a sequence of instructions that will automate the performance of a task on a computer, often for solving a given problem. The process of programming thus often requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Contents

1. History
2. Modern programming
    1. Quality requirements
    2. Readability of source code
    3. Algorithmic complexity
        1. Chess algorithms as an example
3. Programming languages
4. Programmers

History

Programmable devices have existed at least as far back as 1206 AD, when the automata of Al-Jazari were programmable, via pegs and cams, to play various rhythms and drum patterns; and the 1801 Jacquard loom could produce entirely different weaves by changing the "program" - a series of pasteboard cards with holes punched in them.

However, the first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine. Women would continue to dominate the field of computer programming until the mid 1960s.

In the 1880s Herman Hollerith invented the concept of storing data in machine-readable form. Later a control panel (plugboard) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way; as were the first electronic computers. However, with the concept of the stored-program computers introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory.

Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Assembly languages were soon developed that let the programmer specify instruction in a text format, (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses. However, because an assembly language is little more than a different notation for a machine language, any two machines with different instruction sets also have different assembly languages. Kathleen Booth created one of the first Assembly languages in 1950 for various computers at Birkbeck College.

Modern programming

Quality requirements

Whatever the approach to development may be, the final program must satisfy some fundamental properties. The following properties are among the most important:

- Reliability: how often the results of a program are correct. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management (e.g., buffer overflows and race conditions) and logic errors (such as division by zero or off-by-one errors).
- Robustness: how well a program anticipates problems due to errors (not bugs). This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, user error, and unexpected power outages.
- Usability: the ergonomics of a program: the ease with which a person can use the program for its intended purpose or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues. This involves a wide range of textual, graphical and sometimes hardware elements that improve the clarity, intuitiveness, cohesiveness and completeness of a program's user interface.
- Portability: the range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run. This depends on differences in the programming facilities provided by the different platforms, including hardware and operating system resources, expected behavior of the hardware and operating system, and availability of platform specific compilers (and sometimes libraries) for the language of the source code.
- Maintainability: the ease with which a program can be modified by its present or future developers in order to make improvements or customizations, fix bugs and security holes, or adapt it to new environments. Good practices[24] during initial development make the difference in this regard. This quality may not be directly apparent to the end user but it can significantly affect the fate of a program over the long term.
- Efficiency/performance: Measure of system resources a program consumes (processor time, memory space, slow devices such as disks, network bandwidth and to some extent even user interaction): the less, the better. This also includes careful management of resources, for example cleaning up temporary files and eliminating memory leaks.

Readability of source code

In computer programming, readability refers to the ease with which a human reader can comprehend the purpose, control flow, and operation of source code. It affects the aspects of quality above, including portability, usability and most importantly maintainability.

Readability is important because programmers spend the majority of their time reading, trying to understand and modifying existing source code, rather than writing new source code. Unreadable code often leads to bugs, inefficiencies, and duplicated code. A study[25] found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.

Algorithmic complexity

The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problem. For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances.

Chess algorithms as an example

"Programming a Computer for Playing Chess" was a 1950 paper that evaluated a "minimax" algorithm that is part of the history of algorithmic complexity; a course on IBM's Deep Blue (chess computer) is part of the computer science curriculum at Stanford University.[27]

Programming languages

Different programming languages support different styles of programming (called programming paradigms). The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Ideally, the programming language best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. It is usually easier to code in "high-level" languages than in "low-level" ones.

Programmers

Computer programmers are those who write computer software. Their jobs usually involve:

- Coding
- Debugging
- Documentation
- Integration
- Maintenance
- Requirements analysis
- Software architecture
- Software testing
- Specification