Programming Fundamentals

Computer programming (often shortened to programming) is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation (commonly referred to as coding) of algorithms in a target programming language

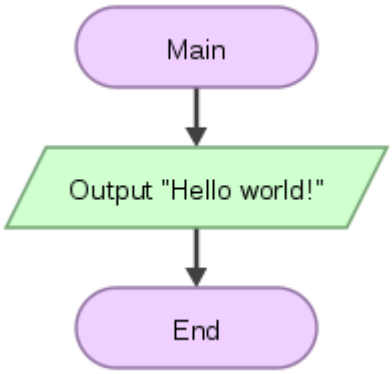
This course comprises nine lessons on the fundamentals of computer programming. Each lesson includes a combination of Wikibooks, Wikipedia, and Internet-based readings, YouTube videos, and hands-on, interactive learning activities. Examples are provided using flowcharts, pseudocode, and a wide variety of computer programming languages.

This entire Wikiversity course can be downloaded in book form by selecting Download Learning Guide in the sidebar. The corresponding Wikipedia reading collection can be downloaded in book form by selecting Download Reading Guide.

Preparation

This is a second-semester, college-level course. Learners should already be familiar with introductory computer concepts and have advanced or proficient-level computer skills. Learners need to have access to the internet readily available to them.

Programming Fundamentals/Introduction



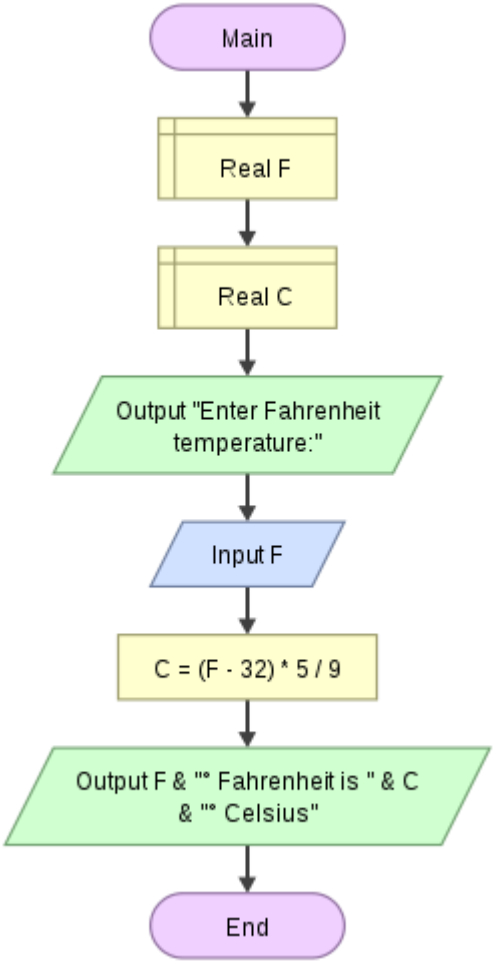
This lesson introduces computer programming, flowcharts, pseudocode, and integrated development environments (IDEs). These will help you to learn a chosen programming language as well as help you learn to write your own programs. Each of the programs are essential to helping you take the steps needed to go further in your field.

Objectives and Skills

Objectives and skills for this lesson include:

- Understand fundamental computer programming concepts.
- Use a flowchart to describe a simple process.
- Use pseudocode to describe a simple process.
- Use an online IDE to edit and test simple programs.
- Understand available compilers/interpreters and IDEs for a selected programming language.

Programming Fundamentals/Variables

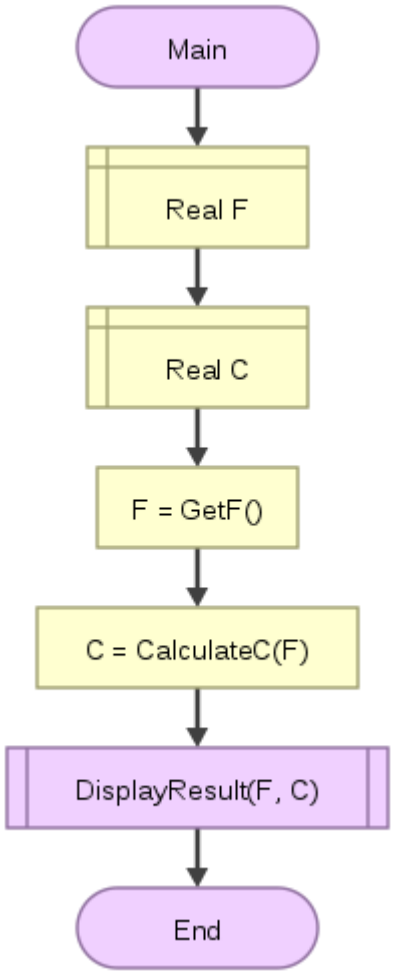


This lesson introduces variables, constants, data types, expressions, statements, and order of operations.

Objectives and Skills

- Understand variables and constants.
- Use integer, floating-point, and string data types appropriately.
- Use expressions and statements to assign values to variables.
- Understand the order of operations for arithmetic and logical operators.

Programming Fundamentals/Functions



This lesson introduces functions. A function is a block of organized code that is used to perform a single task. They provide better modularity for your application and reuse-ability. Depending on the programming language, a function may be called a subroutine, a procedure, a routine, a method, or a subprogram. The generic term, callable unit, is sometimes used. Using functions can allow you to be able to keep your code clean and organized, making it easy to read, and allows the debugging process to be easier.[1]

Objectives and Skills

Objectives and skills for this lesson include:

- Understand the benefits of modular programming
- Understand functions, passed parameters, and return values

- Understand variable scope
- Use functions to implement program functionality
- Use local variables, passed parameters, and return values
- Apply standard coding style to source code

Examples

- Flowchart
- Pseudocode
- Block
- BASIC
- C
- C++
- C#
- Clojure
- COBOL
- Fortran
- Go
- Java
- JavaScript
- Lua
- Perl
- PHP
- PowerShell
- Python3
- Ruby
- Swift
- VB.NET

Activities

Complete the following activities using a flowchart tool, pseudocode, or your selected programming language. Use separate functions for input, processing, and output. Avoid global variables by passing parameters and returning results.

1. Create a program to prompt the user for hours worked per week and rate per hour and then calculate and display their weekly, monthly, and annual gross pay (hours * rate). Base monthly and annual calculations on 12 months per year and 52 weeks per year.[2]
2. Create a program that asks the user how old they are in years, and then calculate and display their approximate age in months, days, hours, and seconds. For example, a person 1 year old is 12 months old, 365 days old, etc.
3. Review MathsIsFun: US Standard Lengths. Create a program that asks the user for a distance in miles, and then calculate and display the distance in yards, feet, and inches, or ask the user for a distance in miles, and then calculate and display the distance in kilometers, meters, and centimeters.
4. Review MathsIsFun: Area of Plane Shapes. Create a program that asks the user for the dimensions of different shapes and then calculate and display the area of the shapes. Do not include shape choices. That will come later. For now, just include multiple shape calculations in sequence.
5. Create a program that calculates the area of a room to determine the amount of floor covering required. The room is rectangular with the dimensions measured in feet with decimal fractions. The output needs to be in square yards. There are 3 linear feet (9 square feet) to a yard.[3]
6. Create a program that helps the user determine how much paint is required to paint a room and how much it will cost. Ask the user for the length, width, and height of a room, the price of a gallon of paint, and the number of square feet that a gallon of paint will cover. Calculate the total area of the four walls as `2 * length * height + 2 * width * height` Calculate the number of gallons as: `total area / square feet per gallon` Note: You must round up to the next full gallon. To round up, add 0.9999 and then convert the resulting value to an

integer. Calculate the total cost of the paint as: `gallons * price per gallon`.[4]

7. Review Wikipedia: Aging in dogs. Create a program to prompt the user for the name of their dog and its age in human years. Calculate and display the age of their dog in dog years, based on the popular myth that one human year equals seven dog years. Be sure to include the dog's name in the output, such as:

   ```
   Spike is 14 years old in dog years.
   ```

Lesson Summary

- Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.[5]
- The hierarchy chart shows the relationship between various models. They are created by the programmer to help document a program.[6]
- Functions allow you to break down programs into smaller, simpler, programs or "blocks" making it easier for the user to test and work on. Functions also allow the user to keep their programs organized and easy to read.[*source?*] A function acts as a miniature program, with its own input, processing, and output.[7]
- The scope is the area of the program where an item that has an identifier name is recognized. It is an important concept for modularization. A Scope can be two types of a Global scope and a Local scope. Global scope occurs when a variable is "defined outside of the function". The local scope, from the other hand, is defined "inside of the function" and exist only until the function completes its task. [8]
- A good rule of thumb for identifiers in procedural programs is to use verb-noun combinations for function identifiers and use a noun or adjective-noun combinations for constant and variable identifiers. If a function name requires two verbs or

two nouns to fully describe the function, it should probably be split into separate functions.[9]

- Programming style is a set of rules or guidelines of writing the code for a computer program. Almost all languages have their own set of guidelines that allow programmers to code efficiently. Following a programming style of a particular language will help programmers to read and understand source code confirming to the style set and help to avoid introducing errors. Code that is written well should have appropriate spacing and proper indentations and be free from grammar and spelling errors.

Key Terms

argument

In programming, a value that is passed between programs, subroutines or functions; which are provided as an input to a function. Arguments are independent items, or variables, that contain data or codes. When an argument is used to customize a program for a user, it is typically called a "parameter."

call-by-reference

Arguments are passed to the subroutine by direct reference, typically using the argument's address. Which can be modified by called functions.

call-by-value

Arguments are evaluated and a copy of the value is passed to the subroutine. Which cannot be modified by called functions.

function

A section of a program designed to perform a specific procedure or task.

function header

The header includes the name of the function and tells us (and the compiler) what type of data it expects to receive (the parameters) and the type of data it will return (return value type) to the calling function or program.

identifier name

The name given by the programmer to identify a function or other program items such as variables

modularization

The ability to group code into a unit, most often being functions, that can be used as independent and self-contained sub-programs withing the main program.

parameter

In computer programming, a parameter is a value that is passed into a function

return statement

Stops the function and returns a variable to the call location.[20]

return value

Return value is a variable or other information coming back from the subroutine.

scope

Variables can only effect areas in which they are defined, their reach by default is local to the function in which they are defined.

subroutine

In computer programming, a subroutine is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

void

A data type that represents a return of no value.[