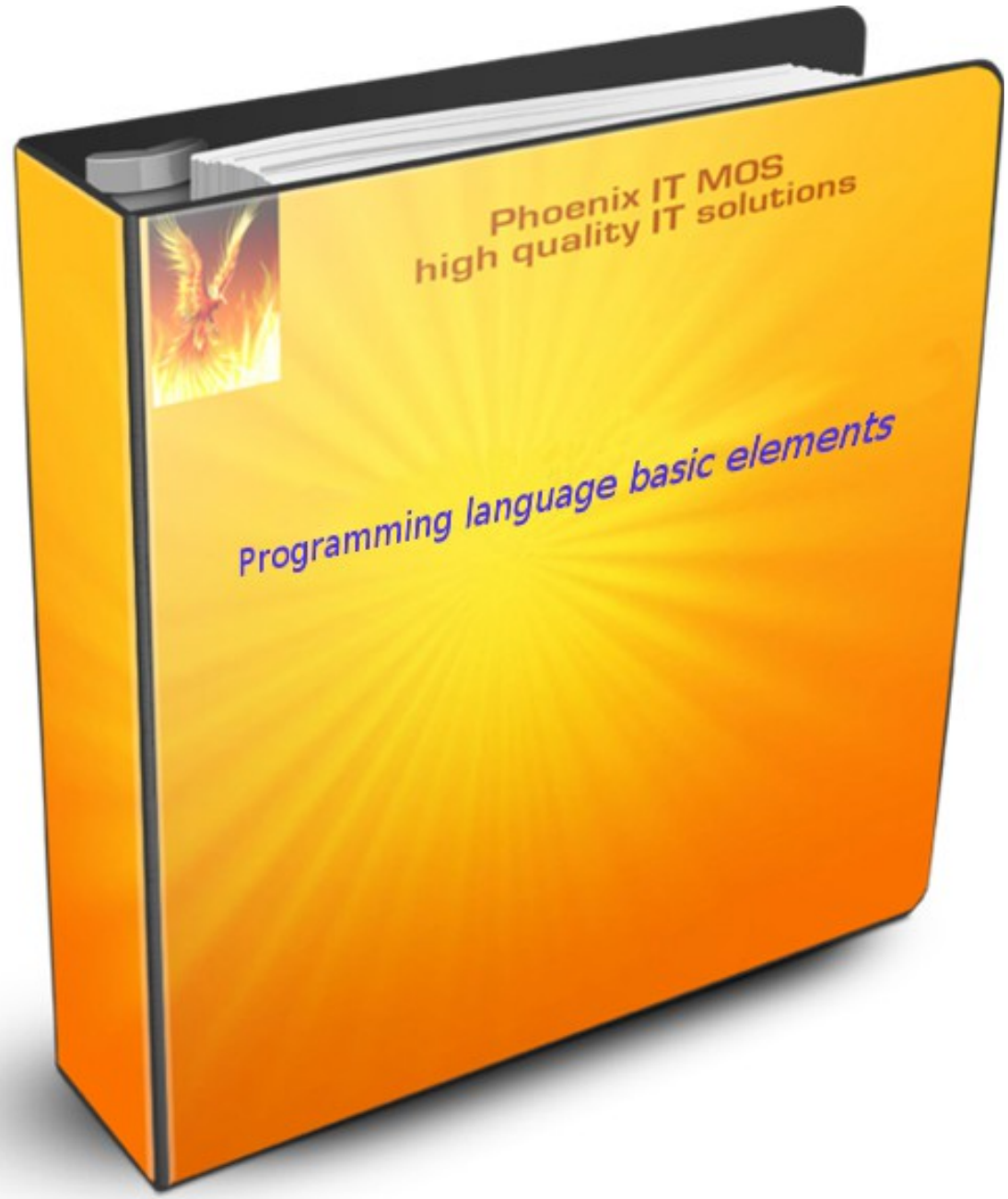
Phoenix IT MOS
high quality IT solutions
Programming language basic elements

Programming language basic elements by Marc Oscar Schwager

Background of this document + what it is

I put this information together in order to have a supplement to the school program in computer science for my son.

This document contains a guideline for teachers.

This course gives a general introduction to the basic elements of a high level programming language. Warning : I use ONLY pseudo code in this document !

Prerequisites

Computer Crash Course (CCC)

The Computer Crash Course contains a guideline for teachers in basic computer knowledge.
It can also be used as an introduction for beginners. You have to follow the mentioned links and read additional information concerning the keywords in the CCC document, otherwise you won't understand this course.

Further reading

Programming introduction

The Programming introduction is a very short "jump" into programming at a basic level. This document will give you an idea of writing a shell-script and programming in a high level programming language.

Python examples

The Python examples should help non programmers to understand Python programming language basics. This document is a supplement to the Programming introduction document.

Table of Contents

Variables

At the core of any program are variables. Variables are "boxes" for storing information.
Every variable has :

- a name called the variable name (example : a)
- a content/value (example a = 15)
- a scope
- a lifetime
- a data type

Scope

The scope of a variable is the part of your program where you are able to refer to and use a variable. The scope of a variable is simply the part of code where you can “see” and use it.

- Local

Local variables are usually variables only accessible in a function. They are not accessible for the code of the program.

- Global

Global variables are accessible in the whole program (and declared outside functions).

Lifetime

The lifetime of a variable is the time period from the creation/definition of the variable until its is destruction. The lifetime of a variable means how long it exists and “lives” in the memory.

The lifetime of a variable ends when it goes out of scope !
As soon as the computer exits a function - we defined a local variable inside - the local variable will be deleted from the memory.
Since a global variable is not contained within any code block, it will only be deleted when the program ends.

Data types

A variable's data type indicates what ***sort of value*** the variable represents (example : integer, string ...).

Dynamic data types : The data type of the variable is determined at run-time.

Static data types : The data type of the variable is determined at compile-time.

integer

An integer is a whole number, there are no digits after a decimal point. (example : 123)

float

A float is a number that may have digits after the decimal place. (example : 123.55)

string

A string is a variable which stores a series of characters.
Examples : "this is a string" or "this 55 is also * a string !"

boolean

Booleans can only have two values like : true or false, 0 or 1, ON or OFF.

Operators

Arithmetic

Addition	c = a + b	
Subtraction	c = a - b	
Multiplication	c = a * b	
Division	c = a / b	
Modulus	c = a % b	c will be the reminder of a divided by b

Comparison

equal	a == b
not equal	a <> b
greater than	a > b
less than	a < b

Logical

AND	a AND b	True if both a and b are true
OR	a OR b	True if either or both a and b are true
XOR	a XOR b	True if either a or b is true, but not both
NOT	! a	True if a is not true. Example : !(4==7) returns true

Assignment

	a = b	a gets the value of b

User input / output

User input means input from the keyboard.

Examples :

```
INPUT a
```

```
SCANF a
```

Output means output on a screen.

Examples :

```
PRINT this will be on the screen
```

```
ECHO this will be also on the screen
```

Control flow structures

Introduction

A control flow structure is a block of programming that analyzes variables and chooses in which direction the program flow goes based on given parameters.
Or in other words : The control flow structures will select the execution of statements depending on whether some conditions are satisfied or not.

The IF condition

IF is a one-time test. IF **a** is true, then do this. IF **a** isn't true the program ignores whatever comes next and carries on with the rest of the code.
You can also provide an alternative with ELSE. If **a** is true, then do this, ELSE do that. It allows you to make decisions in the program.

Example :

```
IF a > b
{
  execute this - if it is true
}
ELSE
{
  execute that - if it's false
}
```

The SWITCH condition

The SWITCH statement is used to perform different actions based on different conditions.

Example :

If **a** is equal to **1**, **2** or **3** execute the corresponding actions **x**,**y** or **z**.
If nothing is true, print a message, then exit the SWITCH.

```
SWITCH a
 {
 CASE a==1
    do action x
    BREAK

 CASE a==2
    do action y
    BREAK

 CASE a==3
    do action z
    BREAK

 DEFAULT
    PRINT "a" is not 1,2 or 3
    EXIT
 }
```

The FOR loop

FOR loops execute a block of code for a specified number of times, or while a specified condition is true.
Abstract :
FOR "start value" to "end value or condition" do something and increment "value"

Example 1 :

```
FOR i=1, 10, i++
 {
  repeat this block of code 10 times
 }
```

Example 2 :

```
FOR i=1, i<=5, i++
 {
  repeat this block of code 5 times
}
```

The WHILE loop

The WHILE loop performs a test each time the loop is performed and continues to loop until a condition has been met.
If the condition never meets, it never stops looping. This can lead to unresponsive programs that crash in an endless loop.

A WHILE loop is a process in which a loop is initiated until a condition has been met. This structure is useful when performing iterative instructions to satisfy a certain parameter (definition from Wikiversity).

Example :

```
a= 0
b=100

WHILE a <> b
{
 a=a+1    increment a
 do something inside this loop until a is equal to b
}
```

Compound data structures

The ARRAY

- Introduction

An array is a special variable, which can store multiple values in it. Values in arrays can be accessed by using the index associated to them. Each element in the array has its own index. Array items are stored contiguously but can be accessed randomly.
They are simple, fast and can be used as basis for more advanced data structures.

- Numeric Arrays : An array with a numeric index.

Example :

Put the values (red, green and blue) in an array. Loop through the array and show the values on the screen.

```
colors = ARRAY(red, green, blue)

FOR i=0, 2, i++)
 {
  PRINT colors[i]
 }
```

- Associative Arrays : An array where each ID key is associated with a value.

Example :

Associate the IDs (sky, tomato, grass) with the values (red, green and blue) and put all together in the array. Loop through the array with FOREACH in order to get each ID and PRINT the values on the screen.

```
colors = ARRAY(sky=>blue, tomato =>red, grass=>green)

FOREACH (ID from colors)
 {
  PRINT colors[ID]
 }
```

Objects

Object-oriented programming may be seen as the design of software using a collection of cooperating objects. In a traditional view a program may be seen as a collection of functions or simply as a list of instructions to the computer.

Object-oriented programming is a method of programming, based on a hierarchy of ***classes*** and well-defined, cooperating ***objects***. Classes and objects are the two main aspects of object oriented programming. Classes can be used to generate multiple instances. Such instances are called objects. An object is a particular instance of a class.

Each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent "machine" with a distinct role. Everything an object can do is represented by its message interface. So you don't have to know anything about what is in the object in order to use it.

Classes are the basic structure of an object (blueprint, template).

Objects have 2 parts :

- Methods — describes how an object can be used or what it can do

  Methods are also called : functions, behaviors, code

- Properties — describes an object's characteristic

  Properties are also called : variables, state information, field, attributes or simply data

Functions

A function is a group of code that has been given a name. By grouping a block of code together and naming it, we can re-use it later and throughout the application without having to rewrite the whole code block again. Not only does it save time and reduces the overall code size of an application, it also means that if there is something wrong, we would only need to change it in one place.

Example:

INPUT **a** receives a number which is your input. The function **square_root** is called with the ***parameter* a** as the function's input. Inside the function, a will be calculated. The ***return value*** (result) of the function will be printed.

```
FUNCTION square_root (a)
 {
   a = a*a
 return (x)
 }

PRINT type in a number
INPUT a
square_root (a)
PRINT the square root is, x
```