Vojtěch Janoušek & Ondrej Lexa:

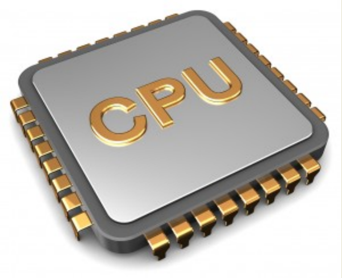
What is a computer program? Principles of algorithmization

Programming languages

Programming language

Is a system for describing computation in machine-readable and, at the same time, human-readable form

- High-level languages
  - the balance is in favor of the human comfort (e.g., *Basic, Pascal, Fortran…*)
- Low-level languages
  - assembly languages and other languages (e.g., *C++*) designed to more closely resemble the computer's processor (CPU) instruction set
  - powerful, quick (used, e.g., in programming operation systems, applications)



- Interpreted
  - each command is executed directly by the interpreter
  - slow; advantages = produce smaller, portable code
- Compiled
  - program has to be previously compiled into the machine language
  - bigger, specific to the given system (CPU); advantage = speed

Programming languages

Scientific programming languages

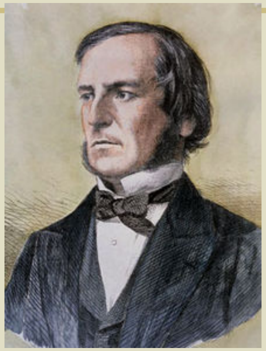
- designed esp. for mathematical computations
- extensive use of matrices
- sophisticated graphical functions
- high-level graphical output
- (statistical tools)
- interpreted, direct vs. batch use (scripting)
- e.g. *ALGOL, FORTRAN, Python, R, Matlab, Julia*

Programming language paradigms

- **Imperative / procedural**
  - command-driven or statement-oriented languages *(e.g., Fortran, C, Pascal)*
  - a program consists of a sequence of statements, the execution of each changes the machine state
- **Functional**
  - everything is viewed as a large function *(e.g., Scheme, LISP)*
- **Declarative** (rule-based)
  - a program consists of a set of rules and it continuously checks whether a particular condition is true
  - if so, the appropriate actions are performed (e.g., *Prolog*)
- **Object-oriented**
  - data structures and algorithms support the abstraction of data
  - the aim is to allow the programmer to use data in a fashion that closely represents their real world use

Many of the widely used programming languages are multi-paradigm, i.e. support object-oriented programming (OOP) to a greater or lesser degree, typically in combination with imperative, procedural programming.

Fundamental data types



George Boole (1815–1864)

Variables

Symbolic name, referring to a storage (memory) which contains some known or unknown quantity of information (a *value*)

Variable types

- **Numeric** (integer, float, real…)
- **Character / text / string**
- **Logical / Boolean**
  only two values are allowed, logical TRUE or FALSE

Many (esp. older) programming languages require first a strict declaration of the variable type (e.g., Pascal)

Modern ones allow an easy conversion, though:
e.g. in R *as.numeric, as.character*
and this is often done automatically
by the interpreter

John von Neumann (1903–1957)

Fundamental data types

Bit = a basic unit of digital information, 0 or 1

Byte = commonly consists of 8 bits

In use are prefixes kilo (10^3 = kB vs. 2^{10}= 1024 = kiB), mega, giga, tera etc.

Decimal positional numeral system (0–10)

Decimal number	**1**	**5**	**1**	**1**	**9**
Multiply by	**10^4**	**10^3**	**10^2**	**10^1**	**10^0**
Decimal result	**10000**	**5000**	**100**	**10**	**9**

Decimal representation of the number 15 119

Binary positional numeral system (0–1)

Binary number	0	0	0	**1**	0	**1**	**1**	0
Multiply by	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal result	0	0	0	**16**	0	**4**	**2**	0

Binary representation of the decimal number 22 is 00010110
Eight bits can express decimal numbers of 0–255

Fundamental data types

Hexadecimal positional numeral system (0–F; A = 10, B = 11...)

Hexadecimal number	**3**	**B**	0	**F**
Multiply by	**16^3**	**16^2**	16^1	**16^0**
Decimal result	**12288**	**2816**	0	**15**

Hexadecimal representation of the number 15119 is #3B0F, &3B0F etc.

Conversion from hexadecimal to binary system

Hex	3				B				0				F			
Bin	0	0	1	1	1	0	1	1	0	0	0	0	1	1	1	1

Just interpret each of the hexadecimal “digits” as four binary ones

Binary representation of the number 15119 (#3B0F) is thus 0011 1011 0000 1111

Fundamental data structures

Types of data structures I.

- o **Homogenous**
  contain elements of the same type
- o **Heterogeneous**
  may contain a mixture of elements of various types

https://en.wikipedia.org/wiki/Data_structure

Types of data structures II.

- **1D (linear)**
  - o **single value (scalar)**
  - o **vector**
    sequence of ordered elements, often of the same type
    these elements can be accessed using an integer index, or by name, specifying which element is required
  - o **categorical**
    can attain only a discrete number of values from a certain dictionary/ look up table (e.g. names of countries)

    in R they are called *factors* and all possible discrete values are *levels*

Fundamental data structures

- **1D (linear) [CONTD.]**
  - **list**
    linear collection of data elements of any type, whereby each has itself a value (can be even other list – lists are recursive structures)
  - **record**
    aggregate data structure, typical in database use
    each record is a value that contains other values, typically in fixed sequence and often indexed by names
- **2D**
  - **matrix / array**
    ordered elements are accessed using an integer index, or by name, to specify which element is required
- **Multi-dimensional**
  - **(multidimensional) array**

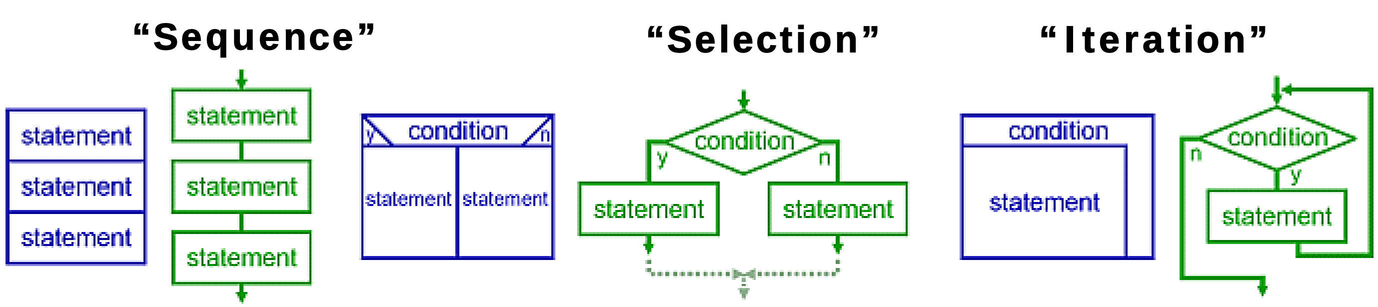
Modern languages allow an easy conversion: e.g. in R *as.matrix, as.vector,…*

This conversion can be even done automatically by the interpreter
(e.g., a matrix with a single row or column becomes a vector in R)

Building blocks of a computer program

Typical computing language contains commands for:

- **Data input**
  from the keyboard, a file, (Internet) connection, or some other device
- **Output**
  on the screen, into a file or to some other device
- **Procedures** that modify data (incl. basic arithmetical operations) **(“Sequence”)**
- **Conditional execution (“Selection”)**
  appropriate sequence of statements is executed only if some condition is fulfilled
- **Loop (“Iteration”)**
  performs some action repeatedly, e.g., given number of times or till something happens (e.g.. variable changes or some event occurs)



Nassi–Shneiderman diagrams (blue) and **flow charts** (green)

Flowcharts

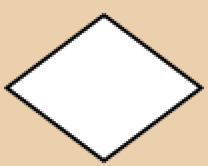
Flowcharts

- Once dominated basic computer science textbooks
- Are used in designing and documenting any complex process
- Consist of various boxes connected by arrows, showing the flow of the process being coded (= flow lines)

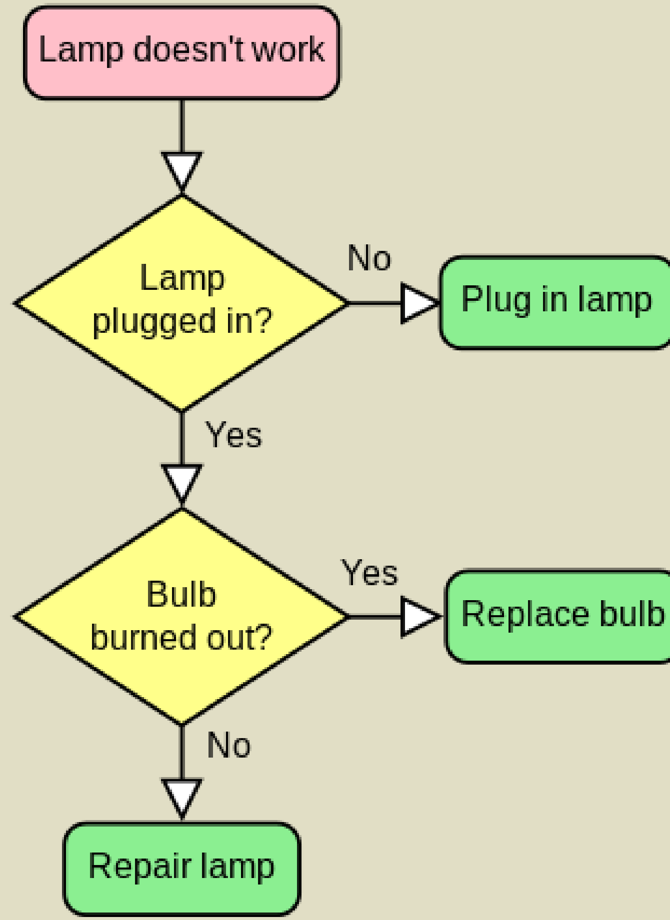
The most common types of boxes are:

– start and end symbols (circle, oval or rounded rectangle)

– processing step, usually called activity (rectangular box)



– conditional or decision (diamond = rhombus) where a decision is necessary, commonly a *Yes/No question* or *TRUE/FALSE test*



https://en.wikipedia.org/wiki/Flowchart

Flowcharts

– Subroutine (procedure)

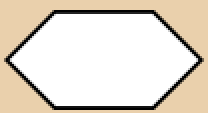
(rectangle with double-struck vertical edges)

- pre-defined complex processing steps
- may be detailed in a separate flowchart

– Input/Output

(parallelogram)

- receiving data or displaying processed data

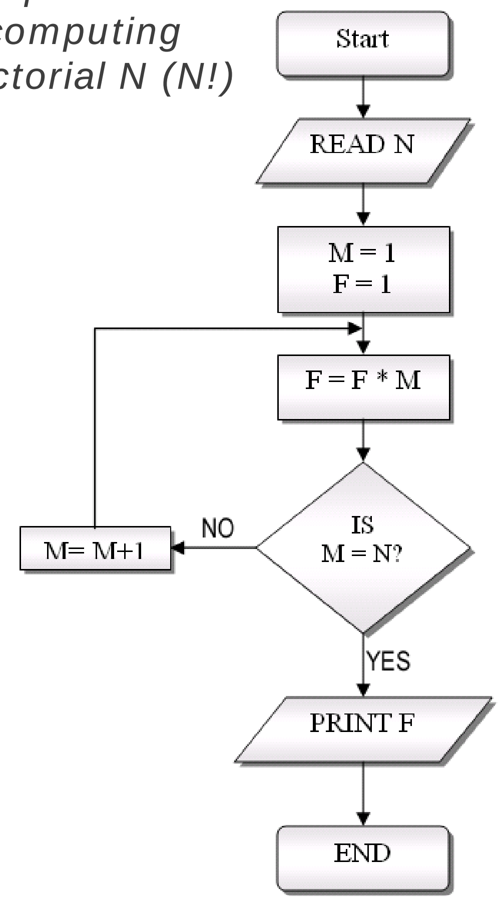


– Prepare conditional

(hexagon)

- operations only initializing a value for a subsequent conditional step or decision
- e.g., in conditional looping

A simple flowchart for computing a factorial N (N!)



https://en.wikipedia.org/wiki/Flowchart

Control of flow

- **Unconditional branches or jumps**
  - e.g., ill-famed *GOTO* of BASIC
  - not recommended as it leads to the "spaghetti code"!
- **Conditional execution**
  - executing a set of statements only if some conditions are met
  - *If –then – (else)*
- **Loops**
  - body of the loop is executed a specified number of times, or once for each of a collection of items, or until some condition is met, or indefinitely
    - Count-controlled loops: *for* loops in many languages
    - Condition-controlled loops: *while, until*
    - Collection-controlled loops:
      allow looping through all elements of a variable (e.g., an array), or all members of a set or collection
      e.g., *for, apply* or *lapply* in R

https://en.wikipedia.org/wiki/Control_flow

Conditional execution – examples in R and Python

if(*condition***)** *expression1* **else** *expression2*

- If condition evaluates to TRUE, expression1 is executed, otherwise expression2 is run.
- In R, complicated commands may be grouped together in braces:

R

```
x <- 6
y <- 0.5
if(x>2 & y<1){
    print(x)
    print(y)
}else{
    cat("Out of range\n")
}
```

Python

```
x = 6
y = 0.5
if x>2 and y<1:
    print(x)
    print(y)
else:
    print("Out of range")
```

Loop – examples in R and Python

for(*variable* **in** *expression1***)** *expression2*

- expression2 is a chunk of R code, usually grouped in braces to be executed repeatedly
- It is done exactly once for each of the values of the control variable, defined by expression1:

R

```
for(f in seq(1,10,by=2)){
    cat("Square root of",f,"is",sqrt(f),"\n")
}
```

Python

```
from numpy import sqrt
for f in range(1, 10, 2):
    print("Square root of", f, "is", sqrt(f))
```

Control of flow

- **Subroutines / procedures**
  - executing a set of distant statements
  - usually after their execution, the flow of control returns back
  - also known as: functions (especially if they return some results) or methods (in Object Oriented Programming, where they are associated with classes)
- **Unconditional halt**
  - prevents any further execution
  - often accompanied by an error message (in error handling)

Functions – examples in R and Python

function.name **<- function(***argument1, argument2, …***)** *expression*

- The `expression` is a chunk of R code, usually grouped in braces
- In order to avoid confusion, the last statement should be `return(var)`, where var is an expression or variable name giving the value(s) to be returned by the function

R

```
stdev <- function(x){
     z <- sqrt(sum((x-mean(x))^2)/length(x))
     return(z)
}
```

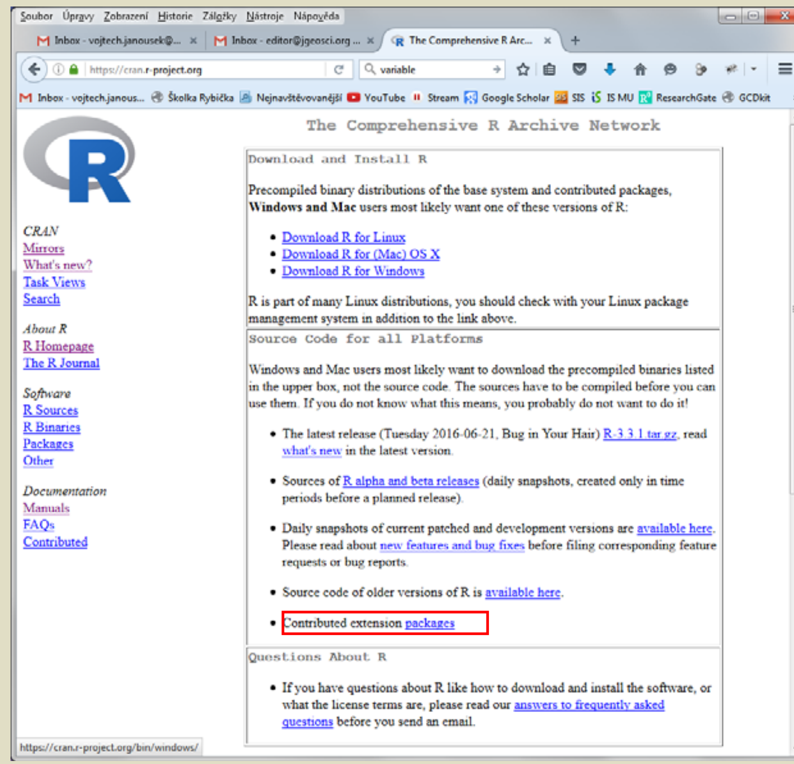
$$S = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n}}$$

Python

```
from numpy import sqrt, sum, mean
def stdev(x):
    z = sqrt(sum((x - mean(x))**2) / len(x))
    return z
```

Packages (R)

- **Packages (“libraries”)**
  - Collections of functions may be provided by external libraries
  - Frequently provide **“toolboxes”** related to a certain task/activity (e.g., R2HTML – output to web pages)
  - Mechanism for distribution of user-defined additions from the Internet community (e.g., CRAN)
  - Often may be **written in some other language** (e.g., calling a C or Fortran code)
  - This can be useful if speed of computation is critical



Scoping and namespaces

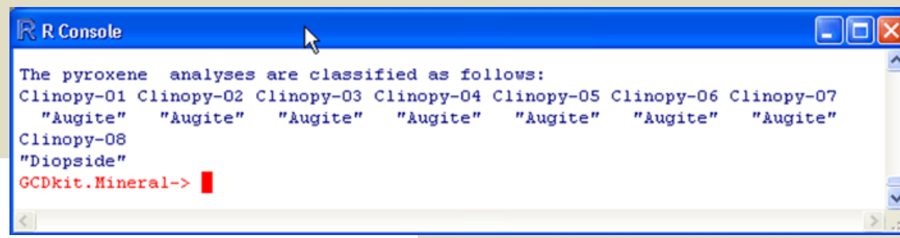
Variable / object types – general definition based on its lifetime

- **Global**
  (always known)
- **Local**
  (particular just to a part of the program, e.g. known only within some procedure)

Object-oriented programming (OOP)

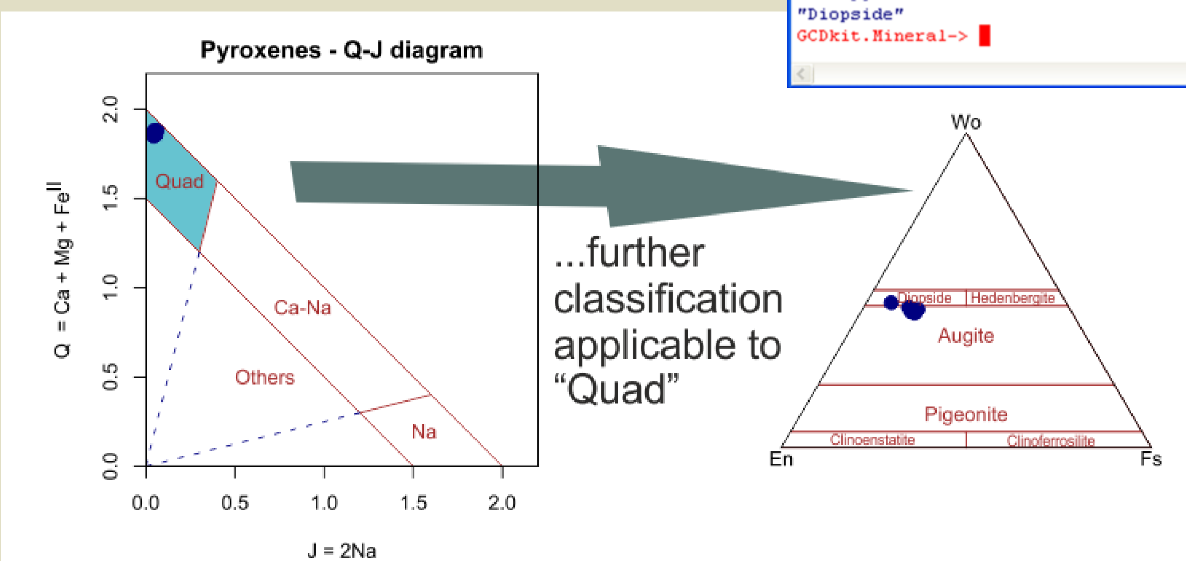
An example – R language (*GCDkit.Mineral*)

- each mineral (*feldspar*, *pyroxene*...) = a subclass of the class *mineral*
- method for formatted printing of the mineral class (for the generic function *print*)
- methods for recalculations to mineral formulae for each mineral species (a subclass)
- methods for IMA classification, based on a pre-defined sequence of binary or ternary plots...



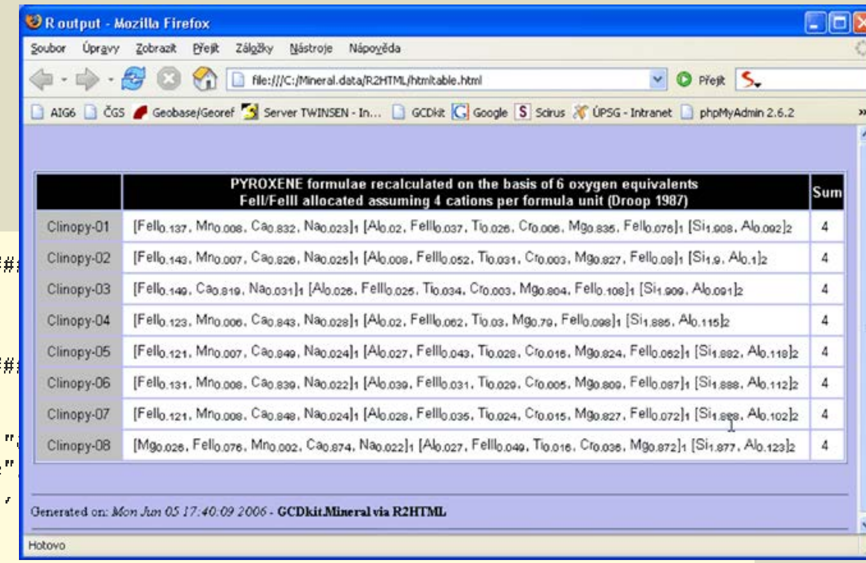
R Console

```
The pyroxene  analyses are classified as follows:
Clinopy-01 Clinopy-02 Clinopy-03 Clinopy-04 Clinopy-05 Clinopy-06 Clinopy-07
  "Augite"   "Augite"   "Augite"   "Augite"   "Augite"   "Augite"   "Augite"
Clinopy-08
"Diopside"
GCDkit.Mineral->
```



Object-oriented programming (OOP)

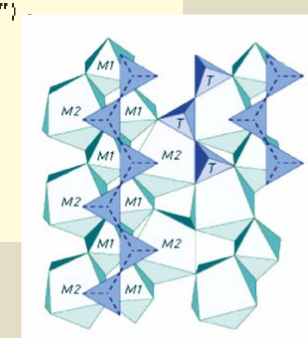
An example – R language (*GCDkit.Mineral*)



	PYROXENE formulae recalculated on the basis of 6 oxygen equivalents FeII/FeIII allocated assuming 4 cations per formula unit (Droop 1987)	Sum
Clinopy-01	$[FeII_{0.137}, Mn_{0.008}, Ca_{0.832}, Na_{0.023}]_1$ $[Al_{0.02}, FeIII_{0.037}, Ti_{0.026}, Cr_{0.006}, Mg_{0.835}, FeII_{0.076}]_1$ $[Si_{1.908}, Al_{0.092}]_2$	4
Clinopy-02	$[FeII_{0.143}, Mn_{0.007}, Ca_{0.826}, Na_{0.025}]_1$ $[Al_{0.008}, FeIII_{0.052}, Ti_{0.031}, Cr_{0.003}, Mg_{0.827}, FeII_{0.08}]_1$ $[Si_{1.9}, Al_{0.1}]_2$	4
Clinopy-03	$[FeII_{0.149}, Ca_{0.819}, Na_{0.031}]_1$ $[Al_{0.026}, FeIII_{0.025}, Ti_{0.034}, Cr_{0.003}, Mg_{0.804}, FeII_{0.108}]_1$ $[Si_{1.909}, Al_{0.091}]_2$	4
Clinopy-04	$[FeII_{0.123}, Mn_{0.006}, Ca_{0.843}, Na_{0.028}]_1$ $[Al_{0.02}, FeIII_{0.062}, Ti_{0.03}, Mg_{0.79}, FeII_{0.098}]_1$ $[Si_{1.885}, Al_{0.115}]_2$	4
Clinopy-05	$[FeII_{0.121}, Mn_{0.007}, Ca_{0.849}, Na_{0.024}]_1$ $[Al_{0.027}, FeIII_{0.043}, Ti_{0.028}, Cr_{0.016}, Mg_{0.824}, FeII_{0.062}]_1$ $[Si_{1.882}, Al_{0.118}]_2$	4
Clinopy-06	$[FeII_{0.131}, Mn_{0.008}, Ca_{0.839}, Na_{0.022}]_1$ $[Al_{0.039}, FeIII_{0.031}, Ti_{0.029}, Cr_{0.005}, Mg_{0.809}, FeII_{0.087}]_1$ $[Si_{1.888}, Al_{0.112}]_2$	4
Clinopy-07	$[FeII_{0.121}, Mn_{0.008}, Ca_{0.848}, Na_{0.024}]_1$ $[Al_{0.028}, FeIII_{0.035}, Ti_{0.024}, Cr_{0.015}, Mg_{0.827}, FeII_{0.072}]_1$ [Si [illegible], $Al_{0.102}]_2$	4
Clinopy-08	$[Mg_{0.026}, FeII_{0.076}, Mn_{0.002}, Ca_{0.874}, Na_{0.022}]_1$ $[Al_{0.027}, FeIII_{0.049}, Ti_{0.016}, Cr_{0.036}, Mg_{0.872}]_1$ $[Si_{1.877}, Al_{0.123}]_2$	4

Generated on: *Mon Jun 05 17:40:09 2006* - **GCDkit.Mineral via R2HTML**

```
#####################################################
#                                PYROXENE
#                           M2[1]M1[1]T[2]O[6]
#
#####################################################
setClass("pyroxene",representation(),contains="mineral",
    prototype(
        abbreviated=c("Px","Cpx","Opx","Di","En","Fs","Hd","
        full=c("pyroxene","Ca clinopyroxene","orthopyroxene",
            "jadeite","pidgeonite","wollastonite","acmite"),
        oxygens=6,
        cations=4,
        iron="Droop",
        # Site allocation according to Morimoto 1988
        atom.names=c("Si","Al","FeIII","Ti","Cr","V","Zr","Zn","Mg","FeII","Mn","Li","Ca","Na","K"),
        sites=list(T=(c("Si","Al","FeIII")),M1=c("Al","FeIII","Ti","Cr","V","Zr","Zn","Mg","FeII","Mn"),
            M2=c("Mg","FeII","Mn","Li","Ca","Na","K")),
        site.sums=c(2,1,NA),
        values.formulae="pyroxene.r",
        values.names=c("FeIII/Fetot","XMg","AlIV/AlVI","AlIV","AlVI"),
        end.member.formulae="pyroxene.end.r",
        end.member.names=c("Jd","Ac","CaTi","CaCr","CaFe","CaTs","Esc","Wo","En","Fs","Ac")
    )
)
```



Object-oriented programming (OOP)

- *objects* contain data in the form of *fields* (e.g., in R termed *slots*)
- the most popular object-oriented languages are class-based, i.e. individual objects are just instances of a given *class*
- the definition of each class includes:
  - initializing function, creating a new object, also setting its default values
  - implementation of a class behavior (*methods* = procedures associated with an object of the given class)

Inheritance

- new (sub-) classes may be defined based on other, more general, simpler ones

Polymorphism

- the functions and operators with the same name can mean different things depending on the class of the operand (e.g., in R *print, plot, summary…*)
- generic functions (used if no specific method is defined for the given class)

Encapsulation

- data and procedures belonging to a particular class are only available for that class

Human readability of the code

– The ease with which a human can comprehend the purpose, flow, and operation of source code

– It may have little to do with the actual code speed/effectivity!

– Human readability can be improved by:

- Using different **indentation styles** (whitespace)
- Extensive use of **comments**, writing detailed **documentation**
- **Decomposition**
  breaking a complex problem into parts that are easier to understand, program and maintain
- **Structured (= modular) programming** which employs:
- block structures
  e.g., in R these are grouped by braces, { }
- subroutines/functions
- conditionals – *if else*
- *for* and *while* loops

https://en.wikipedia.org/wiki/Computer_programming

- Adopting **naming conventions** for objects (variables, functions, classes…)
- Strict and consistent use of naming conventions and structured programming techniques ideally lead to **self-documenting code**

References and further reading

CHAMBERS J.M. (1998) Programming with data. Springer, New York ("Green Book")

CHAMBERS J.M. (2009) Software for data analysis: programming with R. Springer, Berlin

CRAWLEY M.J. (2007) The R book. John Wiley & Sons, Chichester

JANOUŠEK V., MOYEN J.F., MARTIN H., ERBAN V., FARROW C.M. (2016) Geochemical modelling of igneous processes – principles and recipes in R language. Bringing the power of R to a geochemical community. Springer, Berlin

MAINDONALD J., BRAUN J. (2003) Data analysis and graphics using R. Cambridge University Press, Cambridge

LUTZ M. (2007) Learning Python, 3rd Edition, O'Reilly Media

MCKINNEY W. (2012) Python for data analysis, O'Reilly Media

Advanced R, http://adv-r.had.co.nz/

Python Notes, http://www.thomas-cokelaer.info/tutorials/python/

Dive Into Python 3, http://www.diveintopython3.net/

The Python Tutorial, https://docs.python.org/3/tutorial/

The R Project for Statistical Computing, https://www.r-project.org/