Red

Red was created out of the need to get work done, and the tool the author wanted to use, the language of REBOL, had a couple of drawbacks. It was not Open Sourced at that time and it is an interpreted language, what means that it is on average slow compared to a compiled language.

Red, together with its C-level dialect Red/System, provides a language that covers the entire programming space you ever need to program something in. Red is a language heavily based on the language of REBOL. Where Red itself reproduces the flexibility of the REBOL language, the underlying language Red will be built upon, Red/System, covers the more basic needs of programming like C can, being closer to the metal.

Red will be the world's first Full Stack Programming Language. This means that it will be an effective tool to do (almost) any programming task on every level from the metal to the meta without the aid of other stack tools. Furthermore Red will be able to cross-compile Red source code without using any GCC like toolchain from any platform to any other platform. And it will do this all from a binary executable that is supposed to stay under 1 MB.

Ready to learn your first Red?

```
All text before the header will be treated as comment, as long as you avoid using the
word "red" starting with a capital "R" in this pre-header text. This is a temporary
shortcoming of the used lexer but most of the time you start your script or program
with the header itself.
The header of a red script is the capitalized word "red" followed by a
whitespace character followed by a block of square brackets [].
The block of brackets can be filled with useful information about this script or
program: the author's name, the filename, the version, the license, a summary of
what the program does or any other files it needs.
The red/System header is just like the red header, only saying "red/System" and
not "red".

Red []

;this is a commented line

print "Hello Red World"    ; this is another comment

comment {
    This is a multiline comment.
    You just saw the Red version of the "Hello World" program.
}

; Your program's entry point is the first executable code that is found
; no need to restrict this to a 'main' function.

; Valid variable names start with a letter and can contain numbers,
; variables containing only capital A thru F and numbers and ending with 'h' are
; forbidden, because that is how hexadecimal numbers are expressed in Red and
; Red/System.

; assign a value to a variable using a colon ":"
my-name: "Red"
reason-for-using-the-colon: {Assigning values using the colon makes
 the equality sign "=" exclusively usable for comparisons purposes,
 exactly what "=" was intended for in the first place!
 Remember this y = x + 1 and x = 1 => y = 2 stuff from school?
```

```
}
is-this-name-valid?: true

; print output using print, or prin for printing without a newline or linefeed at the
; end of the printed text.

prin " My name is " print my-name
My name is Red

print ["My name is " my-name lf]
My name is Red

; In case you haven't already noticed: statements do NOT end with a semicolon ;-)

;
; Datatypes
;
; If you know Rebol, you probably have noticed it has lots of datatypes. Red
; does not have yet all those types, but as Red want to be close to Rebol it
; will have a lot of datatypes.
; You can recognize types by the exclamation sign at the end. But beware
; names ending with an exclamation sign are allowed.
; Some of the available types are integer! string! block!

; Declaring variables before using them?
; Red knows by itself what variable is best to use for the data you want to use it
; for.
; A variable declaration is not always necessary.
; It is considered good coding practise to declare your variables,
; but it is not forced upon you by Red.
; You can declare a variable and specify its type. a variable's type determines its
; size in bytes.

; Variables of integer! type are usually 4 bytes or 32 bits
my-integer: 0
; Red's integers are signed. No support for unsigned atm but that will come.

; To find out the type of variable use type?
type? my-integer
integer!

; A variable can be initialized using another variable that gets initialized
; at the same time.
i2: 1 + i1: 1

; Arithmetic is straightforward
i1 + i2 ; result 3
i2 - i1 ; result 1
i2 * i1 ; result 2
i1 / i2 ; result 0 (0.5, but truncated towards 0)

; Comparison operators are probably familiar, and unlike in other languages you
; only need a single '=' sign for comparison.
; There is a boolean like type in Red. It has values true and false, but also the
```

```
; values on/off or yes/no can be used

3 = 2 ; result false
3 != 2 ; result true
3 > 2 ; result true
3 < 2 ; result false
2 <= 2 ; result true
2 >= 2 ; result true

;
; Control Structures
;
; if
; Evaluate a block of code if a given condition is true. IF does not return any value,
; so cannot be used in an expression.
if a < 0 [print "a is negative"]

; either
; Evaluate a block of code if a given condition is true, else evaluate an alternative
; block of code. If the last expressions in both blocks have the same type, EITHER can
; be used inside an expression.
either a < 0 [
   either a = 0 [
       msg: "zero"
   ][
       msg: "negative"
   ]
][
   msg: "positive"
]

print ["a is " msg lf]

; There is an alternative way to write this
; (Which is allowed because all code paths return a value of the same type):

msg: either a < 0 [
   either a = 0 [
       "zero"
   ][
       "negative"
   ]
 ][
   "positive"
]
print ["a is " msg lf]

; until
; Loop over a block of code until the condition at end of block, is met.
; UNTIL does not return any value, so it cannot be used in an expression.
c: 5
until [
   prin "o"
   c: c - 1
```

```
   c = 0    ; the condition to end the until loop
]
;   will output:
ooooo
; Note that the loop will always be evaluated at least once, even if the condition is
; not met from the beginning.

; while
; While a given condition is met, evaluate a block of code.
; WHILE does not return any value, so it cannot be used in an expression.
c: 5
while [c > 0][
   prin "o"
   c: c - 1
]
; will output:
ooooo

;
; Functions
;
; function example
twice: function [a [integer!] /one return: [integer!]][
        c: 2
        a: a * c
        either one [a + 1][a]
]
b: 3
print twice b   ; will output 6.

; Import external files with #include and filenames start with a % sign
#include %includefile.red
; Now the functions in the included file can be used too.
```

Further Reading

The main source for information about Red is the Red language homepage.

The source can be found on github.

The Red/System language specification can be found here.

To learn more about Rebol and Red join the chat on Gitter. And if that is not working for you drop a mail to us on the Red mailing list (remove NO_SPAM).

Browse or ask questions on Stack Overflow.

Maybe you want to try Red right away? That is possible on the try Rebol and Red site.

You can also learn Red by learning some Rebol.