Introduction

Rosetta Code is a wiki that offers example programs for many programming languages. JOY is one of those programming languages. From Rosetta Code 26 tasks are taken, one task for each character in the alphabet, and described in this document. The tasks are quite diverse, starting with facts about the language, small test programs for syntactic constructs, up to complete applications. The description in this document solves each task, shows how to incorporate the task in a complete program, and shows the result of running the program.

Links

Rosettacode

This Document

V — variables

The task is to demonstrate variable declaration, initialization, assignment, datatypes, scope, referencing.

JOY does not have variables. Variables are names for locations in memory where values are stored. JOY has names, memory and values but not the combination.

The memory that JOY uses is commonly referred to as "the stack".

Initializing

The JOY stack can be initialized:

```
[] unstack
```

Assignment

Values can be pushed on the stack:

```
42
```

pushes the value 42 of type integer on top of the stack.

Datatypes

Boolean (true, false); Character (‘A …); Integer (42, -42, …); Float (3.14); Set ({0 1 2}); String (“Hello”); List ([0 1 2]); File (returnvalue from fopen); Symbols.

Stack

Calling the stack by name pushes a copy of the stack on the stack. To continue the previous example:

```
stack
```

pushes the list [42] on top of the stack. The stack now contains: [42] 42.

H — hello world / text

The task is to display the string "Goodbye, World!" on a text console.

```
"Goodbye, World!" putchars.
```

C — copy a string

The task is to copy a string.

```
"hello" dup.
```

U — user input / text

The task is to input a string and the integer 75000 from a text console.

```
"Enter a string: " putchars
stdin fgets
"Enter a number: " putchars
stdin fgets 10 strtol.
```

E — execute a system command

The task is to run the "ls" system command.

```
"ls" system.
```

R — rename a file

The task is to rename a file called "input.txt" into "output.txt" and to rename a directory "docs" into "mydocs". This must be done twice: in the working directory as well as in the root directory.

```
"input.txt" "output.txt" frename
"/input.txt" /output.txt" frename
"docs" "mydocs" frename
"/docs" "/mydocs" frename.
```

D — date format

The task is to display the current date in the formats "2007-11-10" and "Sunday, November 10, 2007".

```
time localtime [[0 at 'd 4 4 format] ["-"] [1 at 'd 2 2 format] ["-"] [2 at 'd 2 2 format]] [i] map
[putchars] step '\n putch pop.

DEFINE weekdays == [Monday Tuesday Wednesday Thursday Friday Saturday Sunday];
```

```
      months == [January February March April May June July August September October
November December].

time localtime [[8 at pred weekdays of name] [", "] [1 at pred months of name] [" "]
[2 at 'd 1 1 format] [", "] [0 at 'd 4 4 format] ] [i] map [putchars] step '\n putch pop.
```

X — XML output

The task is to take a list of character names and a list of remarks and produce output like the following example:

```
<CharacterRemarks>
<Character name="April">Bubbly: I'm &gt; Tam and &lt;= Emily</Character>
<Character name="Tam O'Shanter">Burns: "When chapman billies leave the street
..."</Character>
<Character name="Emily">Short &amp; shrift</Character>
</CharacterRemarks>
```

```
DEFINE subst ==
[[['< "&lt;"  putchars]
  ['> "&gt;"  putchars]
  ['& "&amp;" putchars]
  [putch]] case] step;

XMLOutput ==
"<CharacterRemarks>\n" putchars
["<Character name=\"" putchars uncons swap putchars "\">" putchars first subst
"</Character>\n" putchars] step
"</CharacterRemarks>\n" putchars.

[["April" "Bubbly: I'm > Tam and <= Emily"]
 ["Tam O'Shanter" "Burns: \"When chapman billies leave the street ...\""]
 ["Emily" "Short & shrift"]] XMLOutput.
```

Q — quine

The task is to write a program that outputs its own source code.

```
"dup put putchars 10 putch." dup put putchars 10 putch.
```

I — integer sequence

The task is to output all integers, starting with 1.

```
1 [0 >] [dup put succ] while pop.
```

L — loops/infinite

The task is to output "SPAM" followed by a newline in an infinite loop.

```
DEFINE loop == [true []] dip while.
["SPAM\n" putchars] loop.
```

G — greatest common divisor

The task is to find the greatest common divisor of two integers.

```
DEFINE gcd == [0 >] [dup rollup rem] while pop.
```

P — primality by trial division

The task is to tell whether a given integer is prime.

```
DEFINE prime == 2
            [[dup * >] nullary [rem 0 >] dip and]
            [ succ ]
            while
            dup * <.
```

N — number names

The task is to spell out a number in English.

```
DEFINE units ==
["zero" "one" "two" "three" "four" "five" "six" "seven" "eight" "nine" "ten" "eleven" "twelve"
"thirteen" "fourteen" "fifteen" "sixteen" "seventeen" "eighteen" "nineteen"];

tens == ["ten" "twenty" "thirty" "forty" "fifty" "sixty" "seventy" "eighty" "ninety"];

convert6 ==
[1000000 <]
[1000 div swap convert " thousand " putchars convert3]
[1000000 div swap convert " million " putchars convert3]
ifte;

convert5 ==
[null] []
[" and " putchars convert]
ifte;

convert4 ==
[1000 <]
[100 div swap units of putchars " hundred" putchars convert5]
[convert6]
ifte;

convert3 ==
[null] []
[32 putch convert]
ifte;

convert2 ==
[100 <]
[10 div swap pred tens of putchars convert3]
[convert4]
ifte;

convert ==
[20 <]
[units of putchars]
[convert2]
ifte.
```

Y — y combinator

The task is to implement the Y combinator and to use it to calculate factorials and Fibonnaci numbers.

```
DEFINE y == [dup cons] swap concat dup cons i;

       fac == [[pop null] [pop succ] [[dup pred] dip i *] ifte] y.
```

J — jensen's device

The task is to use call by name to calculate the 100th harmonic number.

```
100 [0] [[1.0 swap /] dip +] primrec.
```

B — binary digits

The task is to output the binary digits of a non-negative integer.

```
HIDE
  _ == [null] [pop] [2 div swap] [48 + putch] linrec
IN
  int2bin == [null] [48 + putch] [_] ifte '\n putch
END
```

M — matrix transposition

The task is to transpose an arbitrarily sized rectangular matrix.

```
DEFINE transpose == [[null] [true] [[null] some] ifte]
                    [pop []]
                    [[[first] map] [[rest] map] cleave]
                    [cons]
                   linrec.
```

F — fibonacci sequence

The task is to output the Fibonacci sequence.

```
DEFINE fib == [small]
              []
              [pred dup pred]
              [+]
              binrec.
```

S — sorting algotithms/quicksort

The task is to sort an array (or list) using the quicksort algorithm.

```
DEFINE qsort == [small]              # termination condition: 0 or 1 element
                []                   # do nothing
                [uncons [>] split]   # pivot and two lists
                [enconcat]           # insert the pivot after the recursion
                binrec.              # recursion on the two lists
```

A — ackermann function

The task is to calculate Ackermann(m, n).

```
DEFINE ack == [[[pop null] [popd succ]]
              [[null] [pop pred 1] []]
              [[[dup pred swap] dip pred] [] []]]
              condnestrec.
```

T — towers of hanoi

The task is to solve the Towers of Hanoi problem with recursion.

```
DEFINE hanoi == [[rolldown] infra] dip
                [[[null] [pop pop]]
                [[dup2 [[rotate] infra] dip pred]
                [[dup rest put] dip
                [[swap] infra] dip pred]  []]]
                condnestrec.
```

O — order two numerical lists

The task is to check whether the first of two numerical lists should be ordered before the second or not.

```
DEFINE order ==
[equal] [false]
[[[[size] dip size <=] [[<=] mapr2 true [and] fold]] [i] map i and]
ifte.
```

W — write float arrays to a text file

The task is to write two floating point arrays to a text file, in two columns. Each column has its own precision.

```
DEFINE write-floats ==
['g 0] [formatf] enconcat map rollup
['g 0] [formatf] enconcat map swap zip
"filename" "w" fopen swap
[[fputchars] 9 fputch] step 10 fputch] step
fclose.

[1.0 2.0 3.0 1e11] 3
[1.0 1.41421356 1.73205080 316227.7660168] 5
write-floats.
```

K — knuth shuffle

The task is to create a random permutation of an array.

```
DEFINE knuth-shuffle ==

(* Take the size of the array (without destroying it) *)
dup dup size

(* Generate a list of as many random numbers *)
[rand] [rem] enconcat map

(* Zip the two lists *)
swap zip

(* Sort according to the new index number *)
[small] [] [uncons unswonsd [first >] split [swons] dip2]
[enconcat] binrec

(* Delete the new index number *)
[second] map.
```

Using knuth-shuffle (file shuffle.joy):

```
(* Sorted array of 21 integers *)
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]
knuth-shuffle.
```

Commandline:

```
joy shuffle.joy
```

Output:

```
usrlib  is loaded
inilib  is loaded
agglib  is loaded
[12 6 8 4 14 18 7 15 1 0 11 13 5 10 16 2 19 17 9 20 3]
```

Z — Zig Zag

The task is to produce a zig-zag array.

```
(*
   From the library.
*)
DEFINE reverse == [] swap shunt;
          shunt    == [swons] step.

(*
   Split according to the parameter given.
*)
DEFINE take-drop  == [dup] swap dup [[] cons [take swap] concat concat] dip []
                           cons concat [drop] concat.

(*
   Take the first of a list of lists.
*)
DEFINE take-first == [] cons 3 [dup] times [dup] swap concat [take [first] map
                          swap dup] concat swap concat [drop swap] concat swap
```

```
                     concat [take [rest] step []] concat swap concat [[cons]
                     times swap concat 1 drop] concat.

DEFINE zigzag ==

(*
   Use take-drop to generate a list of lists.
*)
4 [dup] times 1 swap from-to-list swap pred 1 swap from-to-list reverse concat
swap dup * pred 0 swap from-to-list swap [take-drop i] step [pop list] [cons] while

(*
   The odd numbers must be modified with reverse.
*)
[dup size 2 div popd [1 =] [pop reverse] [pop] ifte] map

(*
   Take the first of the first of n lists.
*)
swap dup take-first [i] cons times pop

(*
   Merge the n separate lists.
*)
[] [pop list] [cons] while

(*
   And print them.
*)
swap dup * pred 'd 1 1 format size succ [] cons 'd swons [1 format putchars]
concat [step '\n putch] cons step.

11 zigzag.
```