Quick-Guide.md

PDF of This Guide

If for some reason the font didn't render properly on this page, or some weird font is loaded, the pdf of this guide is available here.

Variable type

There are 3 types of variable in jawascript

Type	in latin	Description
ꦮꦶꦭꦔꦤ꧀	wilangan	Number, can be integer or floating point
ꦠꦸꦭꦶꦱꦤ꧀	tulisan	String
ꦏꦠꦿꦔꦤ꧀	katrangan	Statement

Number literal has to be written with `꧇` surrounding the number, for example : `꧇꧑꧇` `꧇꧑꧒꧇` `꧇꧔꧒꧇` (1, 12, 42).
String literal has to be written with `꧌` surrounding the string, for example : `꧌ꦲꦶꦏꦶꦠꦸꦭꦶꦱꦤ꧀꧍` (iki tulisan).
Statement is special that is has to be in the form of `value1` `comparison type` `value2`, Statement will be discussed deeper on it's own chapter.

Declaring a variable

To declare variable, use keywords `ꦲꦤ (ana)` or `ꦮꦺꦴꦤ꧀ꦠꦼꦤ꧀ (wonten)` followed by **variable name**, followed by `꧈ꦤꦶꦏꦸ (niku)` or `꧈ꦲꦶꦏꦸ (iku)` and then followed by **variable type**, and always end it with `꧉`. for example :

Example	in latin	Description
ꦲꦤꦲ꧈ꦲꦶꦏꦸꦮꦶꦭꦔꦤ꧀꧉	`ana ꦲ, iku wilangan.`	Declaring variable `ꦲ` as number
ꦮꦺꦴꦤ꧀ꦠꦼꦤ꧀ꦧ꧈ꦤꦶꦏꦸꦠꦸꦭꦶꦱꦤ꧀꧉	`wonten ꦧ, niku tulisan.`	Declaring variable `ꦧ` as string
ꦲꦤꦒ꧈ꦤꦶꦏꦸꦏꦠꦿꦔꦤ꧀꧉	`ana ꦒ, niku katrangan.`	Declaring variable `ꦒ` as statement

you can freely interchange between `ꦲꦤ (ana)` and `ꦮꦺꦴꦤ꧀ꦠꦼꦤ꧀ (wonten)`, also `ꦤꦶꦏꦸ (niku)` and `ꦲꦶꦏꦸ (iku)`.
Notice that in some cases, the letter merge, as in `ꦮꦺꦴꦤ꧀ꦠꦼꦤ꧀ꦧ` the `ꦧ` is merged with `ꦤ꧀` because the rule of aksara jawa where `ꦤ꧀ + ꦧ -> ꦤ꧀ꦧ`. In this case, the variable name will still be recognized as `ꦧ`. To help with this, you can use `꧌` and `꧍` as in `ꦮꦺꦴꦤ꧀ꦠꦼꦤ꧀꧌ꦲ꧍` so that the letter won't merge. But remember, in that cases, the variable will be recognized as `꧌ꦲ꧍` and is different from `ꦲ`.
After declared, variables will have a default value. **Number** variables will be set to `0`, **String** variables will be set to `""` (empty string), while **statement** will be set into `""` `equal to` `""`.

Initializing a variable and setting a value

To initialize variable or set a value to a variable, use keyword `ꦒꦤ꧀ꦠꦶ (ganti)` or `ꦒꦤ꧀ꦠꦺꦴꦱ꧀ (gantos)`, followed by **variable name**, followed by `ꦢꦢꦶ (dadi)`, followed by the *value* to assign, and always end it with `꧉`. For example :

Example	in latin	Description
ꦒꦤ꧀ꦠꦶꦲꦢꦢꦶ꧇꧒꧇꧉	`ganti ꦲ dadi 2`	Setting value 2 to variable `ꦲ` (where `ꦲ` is a number variable)
ꦒꦤ꧀ꦠꦶꦧꦢꦢꦶ꧌ꦲꦶꦏꦶꦠꦸꦭꦶꦱꦤ꧀꧍꧉	`ganti ꦧ dadi "iki tulisan"`	Setting value "iki tulisan" to variable `ꦧ` (string)
ꦒꦤ꧀ꦠꦶꦒꦢꦢꦶꦲꦭꦸꦮꦶꦃꦱꦏ꧇꧒꧇꧉	`ganti ꦒ dadi ꦲ luwih saka 2`	Setting a statement "`ꦲ` is greater than 2" to `ꦒ` (statement)

Initialization and value setting done this way must be done with literals, for example, `ꦒꦤ꧀ꦠꦶ꧌ꦥ꧍ꦢꦢꦶ꧌ꦢ꧍꧉ (ganti ꧌ꦥ꧍ dadi ꧌ꦢ꧍)` is not a valid statement to copy the value from `꧌ꦢ꧍` to `꧌ꦥ꧍`, value copying can be done using variable operator.

Printing

To print a value of an existing variable, use the keyword `ꦠꦸꦭꦶꦱ꧀ (tulis)`, followed by **variable name**, and always end it with `꧉`.
For example `ꦠꦸꦭꦶꦱ꧀꧌ꦥ꧍꧉``(tulis ꧌ꦥ꧍.)` will print the value of `꧌ꦥ꧍`.
You can also print a literal, for example `ꦠꦸꦭꦶꦱ꧀꧊ꦠꦸꦭꦶꦱꦤ꧀꧊꧉` `(tulis "ꦠꦸꦭꦶꦱꦤ꧀").` will print `ꦠꦸꦭꦶꦱꦤ꧀`. And `ꦠꦸꦭꦶꦱ꧀꧇꧑꧒꧇꧉` `(tulis 12.)` will print `12`.
To print a new line you can use `ꦒꦫꦶꦱ꧀ꦲꦚꦂ꧉ (garis anyar)` or `ꦧꦫꦶꦱ꧀ꦲꦚꦂ꧉ (baris anyar)`.

Statement

Statement variable has value in the form of `value1` `comparison type` `value2`. A value could be either a literal or a variable name. The comparison types that are available are :

Comparison Type	in latin	Description
ꦭꦸꦮꦶꦃꦱꦏ	luwih saka	`>` greater than
ꦏꦸꦫꦁꦱꦏ	kurang saka	`<` less than
ꦥꦝꦏꦫꦺꦴ	padha karo	`=` equal to
ꦲꦺꦴꦫ	ora	`¬` not

for example, when variable `ꦒ` is set to the statement "`ꦥ` is greater than 2", it'll evaluate to either `true` or `false` depending on the value of `ꦥ`. String comparison are also possible.
Out of 4 comparison types that are available, `ꦲꦺꦴꦫ (ora)` is a bit different that it only accept one argument. So let's say we have a statement variable `ꦥ (pa)` and we wan't to negate it with variable `ꦝ (dha)` i.e. `ꦝ := ¬ꦥ`, we can set `ꦝ` with `ꦒꦤ꧀ꦠꦶꦝꦢꦢꦶꦲꦺꦴꦫꦥ (ganti ꦝ dadi ora ꦥ)`.
There are two default statement variable that have predetermined value, that is
`ꦧꦼꦤꦼꦂ (bener)` which return `true`, and
`ꦱꦭꦃ (salah)` which return `false`.

Variable Operation

You can't do explicit math operation in this language, but you can modify variable using operator. Operator that are available are :

Operator	in latin	Description
ꦠꦩ꧀ꦧꦃ	tambah	add
ꦏꦸꦫꦔꦶ	kurangi	subtract
ꦥꦶꦁ	ping	multiply
ꦥꦫ	para	divide
ꦠꦸꦫꦲꦺꦪꦺꦤ꧀ꦢꦶꦥꦫ	turahé yén dipara	modulo
ꦥꦝꦏ꧀ꦏꦺ	padhakké	set into

To operate on a variable, use keyword `ꦒꦤ꧀ꦠꦶ (ganti)`, followed by **variable name**, followed by `꧈`, followed by **operator**, followed by **operand**, and always end it with `꧉`.
these operator can be called to an existing variable, and will modify it's value. All the operator takes one argument as the operand. Operand can be literal, but also can be variable name. All operation works on **number**, **string** can only use `ꦠꦩ꧀ꦧꦃ (tambah)` and `ꦥꦝꦏ꧀ꦏꦺ (padhakké)`, while **statement** can only use `ꦥꦝꦏ꧀ꦏꦺ padhakké`
for example, if we have variable `ꦥ` with a value of `꧇꧕꧇` `5`, when we call `ꦒꦤ꧀ꦠꦶꦥ꧈ꦠꦩ꧀ꦧꦃ꧇꧒꧇꧉` `(ganti ꦥ, tambah 2)`, the value of `ꦥ` will become `5+2 = 7`.

Loops and conditional

Currently, this language only support while loop, which can be called using the keyword `ꦤꦭꦶꦏꦠꦏ꧀ꦱꦶꦃ (nalika taksih)`, followed by **statement variable**, and always end it with `꧉`. After that, close the loop with the keyword `ꦢꦶꦭꦏꦺꦴꦏꦏꦺ꧉ (dilakoké)`. For example to print the number from `1` to `10` we can write

```
ꦲꦤ꧌ꦲ꧍꧈ꦲꦶꦏꦸꦮꦶꦭꦔꦤ꧀꧉
ꦲꦤ꧌ꦤ꧍꧈ꦲꦶꦏꦸꦏꦠꦿꦔꦤ꧀꧉
ꦒꦤ꧀ꦠꦶ꧌ꦲ꧍ꦢꦢꦶ꧇꧑꧇꧉
ꦒꦤ꧀ꦠꦶ꧌ꦤ꧍ꦢꦢꦶ꧌ꦲ꧍ꦏꦸꦫꦁꦱꦏ꧇꧑꧑꧇꧉
ꦤꦭꦶꦏꦠꦏ꧀ꦱꦶꦃ꧌ꦤ꧍꧉
        ꦠꦸꦭꦶꦱ꧀꧌ꦲ꧍꧉ꦠꦸꦭꦶꦱ꧀ ꧌꧉
        ꦒꦤ꧀ꦠꦶ꧌ꦲ꧍꧈ꦠꦩ꧀ꦧꦃ꧇꧑꧇꧉
ꦢꦶꦭꦏꦺꦴꦏꦏꦺ꧉
```

explanation

Code	in latin	pseudocode
`ꦲꦤ꧌ꦲ꧍꧈ꦲꦶꦏꦸꦮꦶꦭꦔꦤ꧀꧉`	`ana (ꦲ), iku wilangan.`	`var (ꦲ) : numeral`
`ꦲꦤ꧌ꦤ꧍꧈ꦲꦶꦏꦸꦏꦠꦿꦔꦤ꧀꧉`	`ana (ꦤ), iku katrangan.`	`var (ꦤ) : statement`
`ꦒꦤ꧀ꦠꦶ꧌ꦲ꧍ꦢꦢꦶ꧇꧑꧇꧉`	`ganti (ꦲ) dadi 1.`	`(ꦲ) := 1`
`ꦒꦤ꧀ꦠꦶ꧌ꦤ꧍ꦢꦢꦶ꧌ꦲ꧍ꦏꦸꦫꦁꦱꦏ꧇꧑꧑꧇꧉`	`ganti (ꦤ) dadi (ꦲ) kurang saka 11.`	`(ꦤ) := (ꦲ) < 11`
`ꦤꦭꦶꦏꦠꦏ꧀ꦱꦶꦃ꧌ꦤ꧍꧉`	`nalika taksih (ꦤ).`	`while (ꦤ) {`
`ꦠꦸꦭꦶꦱ꧀꧌ꦲ꧍꧉`	`tulis (ꦲ).`	`print (ꦲ)`
`ꦠꦸꦭꦶꦱ꧀ ꧌꧉`	`tulis " ".`	`print " "`
`ꦒꦤ꧀ꦠꦶ꧌ꦲ꧍꧈ꦠꦩ꧀ꦧꦃ꧇꧑꧇꧉`	`ganti (ꦲ), tambah 1.`	`(ꦲ) := ꦲ+1`
`ꦢꦶꦭꦏꦺꦴꦏꦏꦺ꧉`	`dilakokake`	`}`

Loop can be exited with the keyword `ꦫꦩ꧀ꦥꦸꦁ꧉ (rampung.)`.

Keyword	in latin	Description
ꦫꦩ꧀ꦥꦸꦁ	rampung	break out of loop

Conditional if can be constructed using while loop and breaking the loop after. For example

```
ꦤꦭꦶꦏꦠꦏ꧀ꦱꦶꦃ꧌ꦧ꧍꧉
        ...instructions..
        ꦫꦩ꧀ꦥꦸꦁ꧉
ꦢꦶꦭꦏꦺꦴꦏꦏꦺ꧉
```