Walley Language ---- *version 0.03*

Instruction version 0.01

==========

variable type:

- **string -- only “”**
- **bool -- true false**
- **table -- [1,2,3,4] or [a=12,b=14]**
  - **no list type anymore**
  - ***attention: start_index is 0 not 1 like lua***
- **none -- none**
- **number -- 1, 1.3, 1.6**
- **function -- x = def (param1) then statements end**

==========

about the initiation of table

```
x=[ 1 , 2 , 3 ]
     0   1   2

x=[a=12,b=13]
x.a ---> 12
x.b ---> 13
x[“a”] --> 12

x=[“a”=12] is invalid, “a” as key is invalid

x=[a=12]
x[“b”]=14 ---> x=[a=12,b=14]
```

==========

keyword:

and break then if elif else end true false
for def if in local none not or return while

==========

annotation:

- **# one line annotation**
- **#~ *statements* ~# multi-lines annotation**

==========

define a function

the first way:

```
def x ( params ) then    # func_name:x  params : params
        statements       # run statements
end                      # end of defining a function
```

transfer the first way to the second way automatically

--

the second way:

```
x = def (params) then  # func_name :x params : params
        statements     # run statements
end                    # end of defining a function
```

--

==========

The third way and the fourth way will not be developed at first

==========

the third way:

```
def  add:param1 andAdd: param2 then
#func_name1 add | func_name2 andAdd
#func_param1 param1|func_param2 param2
            statements     # run statements
end                    # end of defining a function
```

transfer the third way to the fourth way automaticlly

--

the fourth way:

```
add@andAdd=def param1@param2 then
        statements     # run statements
        end            # end of defining a function
```

==========

==========

class

There is no keyword *class* in Walley Language
However, we can define a class by using table

```
math={}         # define a table
#define a class (static) value
math.a=12

# define a class method
math.add=def (param1,param2) then return param1+param2 end

--------- about how to define value and method for instance
person={}

# define an instance value
person.@age=12      # @ means it is instance value

# define an instance method
person.heightAccordingToAge=
        def ( self  , age )  then       # self means it is instance method
                return self.age*10    # self.age here is person.@age=12
         end

we can also define instance value inside instance method
person.init=def (self , age) then
                        self.age=12   # equals ---> person.@age=12
                end
==========
```

==========

if elif else

```
if judge_statements then
        statements
end

if judge_statements then
        statements
elif judge_statements then
        statements
elif judge_statements then
        statements
.... # many elif statements
else    #you dont have to have else in the end
        statements
end
```

==========

for statements

1.

```
for expr, judge_statements, expr then
        statements
end

eg:
for i=0 , i<10, i++ then
        print(i)
end
```

2.

```
for judge_statements,expr then
```

statements
end

eg:
i=12
for i<12,i++ then
    print(i)
end

3.

for values in value then
    statements
end

situation:	values	value
1.	key,value	table
2.	key	table
3.	char	string

==========
while statements

while judge_statements then
    statements
end

==========