Examples

List operations.
Managing queue outside Restaurant.

Multiple keys with same value.

Convert numbers of address into words.
Function that returns first number from 1d-nums() that gets repeated odd times.
Function that removes all mentions of first three members of 1d-list().
Two 1d-nums() in descending order got mixed. Separate them.

Arrange given dict(=1) by sub-key “c”.

Board game!

How old are you?

Email

List operations:

```
var data = [[1, 2], [3, 4]];

        data.push([5, 6]);
        $ data = [[1, 2], [3, 4], [5, 6]];

        data.unshift([-1, 0]);
        $ data = [[-1, 0], [1, 2], [3, 4]];

        var new_data = data.pop();
        $ data = [[1, 2]];
        $ new_data = [3, 4];

        var new_data = data.shift();
        $ data = [[3, 4]];
        $ new_data = [1, 2];


<any(data)>((1, 2), (3, 4))

        .append (5, 6)<data>

        .insert (1), (-1, 0)<data>

        ..get (-1)<any(new_data)>{data}
        .pop<data>

        ..get (1)<any(new_data)>{data}
        .pop (1)<data>
```

Managing queue outside Restaurant:

```
function newList(list, item){
        list.push(item);
        return list.shift(); removes first item of array and returns that first item.
        }

var myList = [1, 2, 3, 4, 5];
console.log(JSON.stringify(myList));
$ [1, 2, 3, 4, 5]
console.log(newList(myList, 6));
$ 1
console.log(JSON.stringify(myList));
$ [2, 3, 4, 5, 6]



<any(list)>(1, 2, 3, 4, 5)

def newlist():
        get global(list);
        get item, none;
        .append (item)<list>
        return .get (1){list};
        .pop (1)<list>

<out>{list}
$ <(1, 2, 3, 4, 5)>
newlist(6: num)
<out>{num}
$ (1)
<out>{list}
$ <(2, 3, 4, 5, 6)>
```

Multiple keys with same value:

```
d = {}

d[1] = d[2] = d[3] = 'yes'
d[4] = 'no'
```

or

```
d1 = {'yes': [1, 2, 3], 'no': [4]}
d = {value: key for key in d1 for value in d1[key]}

print(d)
$ {1: 'yes', 2: 'yes', 3: 'yes', 4: 'no'}
```

Note:

```
d = {}
d[1] = d[2] = d[3] = d
d[4] = False
print(d)
$ {1: {...}, 2: {...}, 3: {...}, 4: False}
```

Note:

d1 = {'yes': [1, 2, 3], 'no': 4}

d = ...

generates TypeError: 'int' Object is not iterable

<any(d)><yes: <(1, 2, 4)> no: (3)>
<dct.list><>

loop:
	: for any(item) in d AND key in keys
	: if item is bit.zero():
		.add (item), (key)<dct.list>
	else:

		loop:
			: for i in item
			: .add (i), (key)<dct.list>

or loop:
	; <dct.list><>
	: for any(item) in d AND key in keys
	: if item is bit.zero():
		.add (item), (key)<dct.list>
	else:
		.add,keys (item), (key)<dct.list>

§ <(1): yes (2): yes (4): yes (3): no>

Note:
<dct.list><>
.add,keys (1, 2, 4), (true)<dct.list> **Here, first selfArgument is nums().**
.add (3), (false)<dct.list> **Here, first selfArgument is int().**

§ <(1): (true) (2): (true) (4): (true) (3): (false)>

Convert numbers of address into words:

(le1) <address><G\/7\, 104> § ~~G/7, 104~~
(le2) <address><1\, 53612> § ~~1, 53612~~
<set.list><1 : One, 2 : Two, 3 : Three, 4 : Four>

loop:
	; <answer><>
	: for word in address
	: <answer> += .fetch (word), (word){set.list}

For **(le1)**, § ~~G/7, One0Four~~
For **(le2)**, § ~~One, 5Three6OneTwo~~

vs

loop:
	; <answer><>
	: for word in address
	: if NOT .fetch (word) .bool{set.list}:
		<answer> += word
	else: **# This statement generates Variable <it>.**
		if answer is not null() AND .get (-1){answer} != space:
			<answer> += space
		<answer> += it
		<answer> += space
finally:
	parent(answer)

For **(le1)**, § ~~G/7, One 0 Four~~**(space)**
For **(le2)**, § ~~One , 5 Three 6 One Two~~**(space)**

.remove,pos (-1)<answer>

..find,all \,<all_pos.list>{answer}
if all_pos.list is not null():
    .lambda,new a, (a-1)<all_pos.list>

    loop:
        : for pos in all_pos.list
        : if .get (pos){answer} = space:
            .remove,pos (pos)<answer>

For **(le1)**, § ~~G/7, One 0 Four~~
For **(le2)**, § ~~One, 5 Three 6 One Two~~

#! Note:
For **(le1)**, at <answer><G\/7\, >,

if .fetch (word) .bool{set.list}: **This statement generates Variable <it><One>.**
    if .get (-1) .bool{answer} AND it != space: **This statement overwrites as <it>{space}.**
        <answer> += space
    Now, <it>{space} doesn't exist.
    <answer> += it ***statement generates NameE;***
    <answer> += space

Function that returns first number from 1d-nums() that gets repeated odd times:

```
from easyA import has_lst

def first_odd_repeat():

    get nums.list if it is nums() AND NOT has_lst(it);

    loop:
        : for num in nums.list
        : if .count (num){nums.list} % 2 != 0:
            return num;
            return;  # statement (1)
    finally:
        return none;


or def first_odd_repeat():

    get nums.list++ if it is nums() AND NOT has_lst(it);

    loop:
        : if .get (1) .count{nums.list} % 2 = 0:
            .get (1) .remove,all<nums.list>
            # or .pop,nis (.get (1) .find,all{nums.list})<nums.list>
        else:
            collect("e")
    until:
        if nums.list is null():
            return none;
    except:
        return .get (1){nums.list};
```

<nums1.list>(2, 4, 4, 6, 4, 2)
<out>{first_odd_repeat(nums1.list)} $ ~~(4)~~
 # Without statement (1), there are five return statements: return 4; return 4; return 6; return 4; return none; So, outOperator prints $ ~~<(4, 4, 6, 4)>~~

<nums2.list>(2, 5, 5, 2)
<out>{first_odd_repeat(nums2.list)} $ ***(A line generated by /b(null))***

Function that removes all mentions of first three members of 1d-list():
(le1) <lst.list>(2, 4, 4, 6, 4, 2)
(le2) <lst.list><Sarah, John, John, Mia>
(le3) <lst.list>(1, 4, 5, 4)

```
from easyA import has_lst

def less_mentions():  # Note: Function is updater only.

    get lst.list as self if it is nums() or names() AND NOT has_lst(it);

    <pos.list><>

    loop:
        : for value in .get (1:3){lst.list}
        : ..find,all (value) .append,nis<pos.list>{lst.list}
        # For (le1), <pos.list>(1, 6, 2, 3, 5, 2, 3, 5)
        # For (le2), <pos.list>(1, 2, 3, 2, 3)
        # For (le3), <pos.list>(1, 2, 4, 3)

    finally:
        .sort<pos.list>
        # For (le1), <pos.list>(1, 2, 2, 3, 3, 5, 5, 6)
        # For (le2), <pos.list>(1, 2, 2, 3, 3)
        # For (le3), <pos.list>(1, 2, 3, 4)
```

```
                .lambda,filter x, (x!=y)<pos.list>
                # For (le1), <pos.list>(1, -, 2, -, 3, -, 5, 6)
                # For (le2), <pos.list>(1, -, 2, -, 3)
                # For (le3), <pos.list>(1, 2, 3, 4)

        .pop,nis (pos.list)<lst.list>

less_mentions(lst.list)
<out>{lst.list}
# For (le1),  $ <(6)>
# For (le2),  $ <Mia>
# For (le3),  $ <>
```

Two 1d-nums() in descending order got mixed. Separate them:

```
<mixed.list>(90, 75, 89, 87, 72, 86.5, 80, 71, 71, 69, 65, 79)

loop:
        : ..lambda,spread x, (y>x)<lambda.list>{mixed.list}
        if lsts(lambda.list, mixed.list):  # This statement never gets implemented.
                collect("f")
        then ..lambda,spread x, (y>x)<lambda.list>{previous.list}
                if lsts(lambda.list, previous.list):
                        collect("f")
        : <previous.list>{lambda.list}  del(lambda.list)

# 90—75<-89—87—72<-86.5—80—71—71—69—65<-79 is mixed.list
# 90—89—89—87—86.5-86.5—80—71—71—69<-79—79 is lambda.list after first loop
# 90—89—89—87—86.5-86.5—80—71—71<-79—79—79 is lambda.list after second loop
# 90—89—89—87—86.5-86.5—80—71<-79—79—79—79 is lambda.list after third loop
# 90—89—89—87—86.5-86.5—80—79—79—79—79—79 is lambda.list after fourth loop
# 90—89—89—87—86.5-86.5—80—79—79—79—79—79 is lambda.list after fifth loop which is same as lambda.list of previous loop
```

<lst1.list><>
<lst2.list><>

loop:
	: for num in mixed.list
	: for rfr in lambda.list
	: if num = rfr:
		.append (num)<lst1.list>
	else:
		.append (num)<lst2.list>
finally:
	del(lambda.list)

(**90**, 75, **89**, **87**, 72, **86.5**, **80**, 71, 71, 69, 65, **79**) is mixed.list
(**90**, 89, **89**, **87**, 86.5, **86.5**, **80**, 79, 79, 79, 79, **79**) is lambda.list

<out>{lst1.list} $ ~~<(90, 89, 87, 86.5, 80, 79)>~~
<out>{lst2.list} $ ~~<(75, 72, 71, 71, 69, 65)>~~

Arrange following dict(=1) by sub-key “c”:

<dct.list><(1): <p: (350) z: (750) c: (110)>
(6): <p: (100) z: (700) c: (900)>
(4): <p: (300) z: (200) c: (130)>>

loop:
; <lst.list><>
; <keys.list><>
: for any(value) in dct.list AND key in keys
: ..get c .append<lst.list>{value}
.append (key)<keys.list>
finally:
parent(lst.list, keys.list)

Here, <lst.list>(110, 900, 130), <keys.list>(1, 6, 4)

sort_also(lst.list, keys.list, reverse= true) **# Unlike sort(), it takes two Variables.**

Now, <lst.list>(900, 130, 110), **<keys.list>(6, 4, 1)**

sort_keys(dct.list, as= keys.list) **# It works with dict(=1) while sort_pos() works with nums(), names(), dict(>1).**

Now, **<dct.list><(6): <...> (4): <...> (1): <...>>**

Board game:

```
<board.list><-, -, -, -, -, -, -, -, ->

def display_board():
        get rfr(board.list);

        loop:
                ; <out><>
                : for item in board.list AND N in len
                : <out>{item/db}
                if N%3 = 0:
                        <out>{repeat(space, 3)/db}
                        <out><'stringify(N-2)' | 'stringify(N-1)' | 'N'>
                else:
                        <out>< | /db>

$
$ -|-|-  1|2|3
$ -|-|-  4|5|6
$ -|-|-  7|8|9
$


def win():
        get rfr(board.list);

        unpack(board.list : one, two, three, four, five, six, seven, eight, nine)

        if one and two and three = "O" or "X"
        OR four and five and six = "O" or "X"
        OR seven and eight and nine = "O" or "X":
                return true;  # for rows
        elif one and four and seven = "O" or "X"
        OR two and five and eight = "O" or "X"
        OR three and six and nine = "O" or "X":
                return true;  # for colums
```

```
        elif one and five and nine = “O” or “X”
        OR three and five and seven = “O” or “X”:
                return true;  # for diagonals
        else:
                return false;



def tie():
        get rfr(board.list);

        if .find \- .bool{board.list}:
                return false;
        else:
                return true;



loop:
        ; <player.list><X : O>
        ; <player><X>
        : display_board()
        <out><‘player’\’s turn.>

        loop:
                : <position>{in} Choose a number from 1-9\:/b{}
                then <position>{in} {}

                : if bool(beint(position)) AND in_range(it, 1, 9):
                        if .get (it){board.list} = “-“:
                                <position>{it}  del(it)
                                collect(“f”)  # to end child loop. Note: use of parent() terminates
                                parent loop as well.
                        else:
                                <out><you can\’t go there\, go again.>
```

```
.replace,pos (position), (player)<board.list>

if win():
        <out><‘player’ won.>
        collect(“f”)
elif tie():
        <out><Tie.>
        collect(“f”)
else:
        ..fetch (player)<player>{player.list}
```

How old are you:

```
import datetime
import easyA

date.current(none : current)  # <current> is Snip.

loop:
        ; <current_year>{get_smpl(current, “yy”)}  # <current_year> is int().
        : <year>{in} In which year you were born? {}
        then <year>{in} Try again \[yyyy\]\: {}
        : if .len{year} != 4 OR NOT bool(beint(year)):
                <out><you have entered wrong format.>
        else:
                <year>{it}  del(it)
                <age>(current_ year-year)
                if in_range(age, 0, 107):
                        <out><you are ‘age’ ‘sp(age, “year”, “years”)’ old.>
                        # you are 0 year old.  you are 2 years old.
                        collect(“f”)
                else:
                        <out><you have entered wrong year.>


date(“yy”, year : birth_date)

loop:
        : <month>{in} In which month you were born? {}
                // t.num()
after:
        <out><‘month’ is wrong format.>
until:
        if month is int() AND in_range(month, 1, 12);


date.add(birth_date, “mm”, month)
```

```
date.math(“a0”, current, birth_date : age)
if age is null():
        <out><you are 0 month old.>  # you are 0 month old.
else:
        dt.design(age, “%ua”)
        get_str(age, “y”, “m” : y_in_str, m_in_str)
        if y_in_str is null():
                <out><you are ‘m_in_str’ old.>  # you are 6 months old.
        elif m_in_str is null():
                <out><you are ‘y_in_str’ old.>  # you are 1 year old.
        else:
                <out><you are ‘y_in_str’\, ‘m_in_str’ old.>  # you are 1 year, 6 months old.

loop:
        ; allowed_days(birth_date : allowed_days)
        : <day>{in} On which day you were born? {}
                // t.num()
        then <day>{in} {}
                // t.num()
        : if day is int() AND in_range(day, 1, allowed_days):
                collect(“f)

date.declare(day, “dd”)
date.join(birth_date, day)

date.math(“a0”, current, birth_date : age)
if age is null():
        <out><you are 0 day old.>  # you are 0 day old.
else:
        dt.design(age, “%ue, %ud, %ub”)
        <out><you are ‘age’ old.>
        # you are 1 year, 6 months, 28 days old.
        # you are 1 year, 6 months old.  you are 1 year, 28 days old.  you are 1 year old.
        # you are 6 months, 28 days old.  you are 6 months old.  you are 28 days old.
```

Email:

```
from selfDesign apply cluster

<out></b Welcome!>

<usernames.list><b>
<passwords.list><abc1>
<messages.list><>

def user_exists():
        get rfr(usernames.list);
        get name, none;

        if .find (name) .bool{usernames.list}:
                return true;
        else:
                return false;



def pass_generator():
        get password as self;
        get none;

        loop:
                : if .len{password} < 4:
                        <out><password must be greater than 3 characters! Try again...>
                elif .count{password} = 0:
                        // <>{.count.}
                                (cluster) “1”
                        <>{}

                        <out><password must contain at least 1 number! Try again...>
                elif .count{password} < 3:
                        // <>{.count.}
                                (cluster) “aA”
                        <>{}
```

```
                <out><password must contain at least 3 letters! Try again...>
            else:
                collect("f")

            <password>{in} Password\: {}



def get_my():
    get rfr(messages.list);

    if messages.list is null():
        get;  # without this, while implementing, function generates FIE;
        return false;
    else:
        get desired_key, rfr(user), none;  # while defining this function, <user> doesn't
        exist in main env. yet. Still, this works.

        loop:
            ; <found>{false}
            : for message.list in reversed(messages.list)
            : for id in range(.len{messages.list}, 1)
            : if .get (desired_key){message.list} = user:
                <found>{true}

                loop:
                    ; <out></b ID\: 'id'>
                    : for value in message.list AND key in keys
                    : if key != desired_key:
                        <out><'key'\: 'value'>  # here, <value> is always str().

        finally:
            if found:
                return true;
            else:
                return false;
```

```
def login():
    get global(usernames.list, passwords.list);
    get none;

    loop:
        : <ans>{in} | 1 = Log-in | 2 = Sign-up |/b{}
        if ans = “1”:
            <login>{true}
            collect(“f”)
        elif ans = “2”:
            <login>{false}
            collect(“f”)
        else:
            <out><‘ans’ is not a valid option.>

    if login:

        loop:
            : <user>{in} Please enter your username\: {}
        after:
            if user = space:
                <out><username can\’t be space...>
            elif .find (space) .bool{user}:
                <out><username can\’t have any spaces...>
            else:
                <out><‘user’ is not registered...>
        until:
            if user_exists(user);  # It works but it isn’t a good practice, as it searches
            <user> within entire <usernames.list>, even if it is space or it contains
            space...

        <out><Hello ‘user’>
        loop:
            ; <tries>(4)
            ; ..get (.find (user){usernames.list})<testpass>{passwords.list}
            : <pass>{in} Please enter your password\: {}
            <tries> -= 1
```

```
        after:
                <out><Invalid password\, 'tries' more 'sp(tries, "attempt", "attempts")'.>
        until:
                if pass = testpass;
                elif tries = 0:
                        <out><Sorry\, please try again later...>
                        return;  # It is end of program!

        <out><Successfully logged into 'user'\'s account.>
        make global(user, pass);

        menu() #is bounce-calling.

else:
        <out><Add a new account...>

        loop:
                : <newuser>{in} Username\: {}
                if newuser = space:
                        <out><username can\'t be space...>
                elif .find (space) .bool{newuser}:
                        <out><username can\'t have any spaces...>
                elif NOT user_exists(newuser):
                        collect("f")
                else:
                        <out><'newuser' is taken\, try another one...>

        <newpass>{in} Password\: {}
        pass_generator(newpass)
        <out><Successfully created new account.>
        .append (newuser)<usernames.list>
        .append (newpass)<passwords.list>
        login() #is self-calling.
        # It requires user to login just after signup.
        # Vs <user>{newuser}  del(newuser)
        # <pass>{newpass}  del(newpass)
        # menu()  It also does login after signup.
```

login() **#is linear-calling.**

```
def menu():
        get global(user, pass);
        get none;

        loop:
                : <main>{in} | 1 = Email | 2 = Manage account | 3 = Sign out |/b{}
                        // t.num()
        after:
                <out><‘main’ is not a valid option.>
        until:
                if main is int():
                        if main = 1:
                                email()
                        elif main = 2:
                                manage()
                        elif main = 3:
                                del global(user, pass);
                                login()


def email():
        get global(messages.list);
        get rfr(user), none;

        loop:
                : <emailer>{in} | 1 = Compose | 2 = Inbox | 3 = Sent | 4 = Go back |/b{}
                        // t.num()  #terminal must be number.
                if emailer is int() AND i_range(emailer, 1, 4):
                        collect(“f”)
                else:
                        <out><‘emailer’ is not a valid option.>
```

```
if emailer = 1:

        loop:
                : <to>{in} To\: {}
                if to = user:
                        <out><can\'t send message to self...>
                elif user_exists(to):
                        collect("f")
                else:
                        <out><'to' is not registered username.>

        <subject>{in} Subject\: {}
                // t.null()  #terminal can be null.

        loop:
                : <mail>{in} Compose\: {}
                        // t.null()
                if mail is null():
                        <out><Error! Email must contain a message.>
                else:
                        collect("f")

        if subject is null():
                <message.list><From: 'user'  To: 'to'  Message: 'mail'>
        else:
                <message.list><From: 'user'  To: 'to'  Subject: 'subject'  Message: 'mail'>
        .append (message.list)*<messages.list>  # * is concept of main exhaust.
        <out><Message Sent!>

elif emailer = 2:
        <out><Loading your Inbox...>
        <inbox><To>
        get_my(inbox : found)
        if NOT found:
                <out><Inbox empty!>
```

```
    elif emailer = 3:
        <out><Loading sent mails...>
        <sent_mails><From>
        get_my(sent_mails : found)
        if NOT found:
            <out><Empty!>
    else:
        <out><Back to main menu...>
        menu()  # *
        return;  # *

    email()  # Since statements with * means email() ends, menu() starts, ~~else;~~ statement
    that represents ~~"option 4 = Go back"~~ never reaches here.


def manage():
    get global(usernames.list, passwords.list, user, pass);
    get none;
    ..find (user)<user_at>{usernames.list}

    loop:
        : <manager>{in} | 1 = Change Username | 2 = Change Password | 3 = Delete Account
        | 4 = Go back |/b{}
        if manager = "4":
            <out><Back to main menu...>
            menu()
            return;
            # manage() ends, menu() starts.
        elif bool(beint(manager)) AND in_range(it, 1, 3):
            <manager>{it}  del(it)
            collect("f")
        else:
            <out><'manager' is not a valid option.>

    if manager = 1:
```

```
        loop:
                : <newuser>{in} Please enter a new username\: {}
                if newuser = space:
                        <out><username can\'t be space...>
                elif .find (space) .bool{newuser}:
                        <out><username can\'t have any spaces...>
                elif newuser = user:
                        <out><'newuser' is your current username!>
                elif user_exists(newuser):
                        <out><'newuser' is already taken!>
                else:
                        .replace,pos (user_at), (newuser)<usernames.list>
                        <user>{newuser}
                        <out><Your username is 'user'.>
                        manage()
                        return;
                        # Current manage() ends, New manage() starts.

elif manager = 2:
        loop:
                ; <tries>(4)
                : <trypass>{in} Please enter your current password\: {}
                <tries> -= 1
        after:
                <out><Invalid password\, 'tries' more 'sp(tries, "attempt", "attempts")'.>
        until:
                if trypass = pass:
                        <newpass>{in} Now enter your new password\: {}
                        pass_generator(newpass)
                        .replace,pos (user_at), (newpass)<passwords.list>
                        <pass>{newpass}
                        <out><Successfully changed your password.>
                        manage()
                elif tries = 0:
                        <out><No more attempts. Try again later...>
                        del global(user, pass);
                        login()
```

```
else:
        loop:
                ; <tries>(4)
                : <trypass>{in} Please enter your password\: {}
                <tries> -= 1
        after:
                <out><Invalid password\, 'tries' more 'sp(tries, "attempt", "attempts")'.>
        until:
                if trypass = pass;
                elif tries = 0:
                        <out><No more attempts. Try again later...>
                        del global(user, pass);
                        login()
                        return;

        <out><This is irreversible. Are you sure you want to delete your account?>

        loop:
                : <choice>{in} | 1 = Yes | 2 = No |/b{}
                then <choice>{in} {}
        after:
                <out><'choice' is not a valid option.>
        until:
                if choice = "1":
                        .pop (user_at)<usernames.list>
                        .pop (user_at)<passwords.list>
                        del global(user, pass);
                        <out><Successfully deleted your account...>
                        login()
                elif choice = "2":
                        manage()
```