
CREATE PROGRAIMING LANGUAGES
WITH PYTHON
(jero98772)

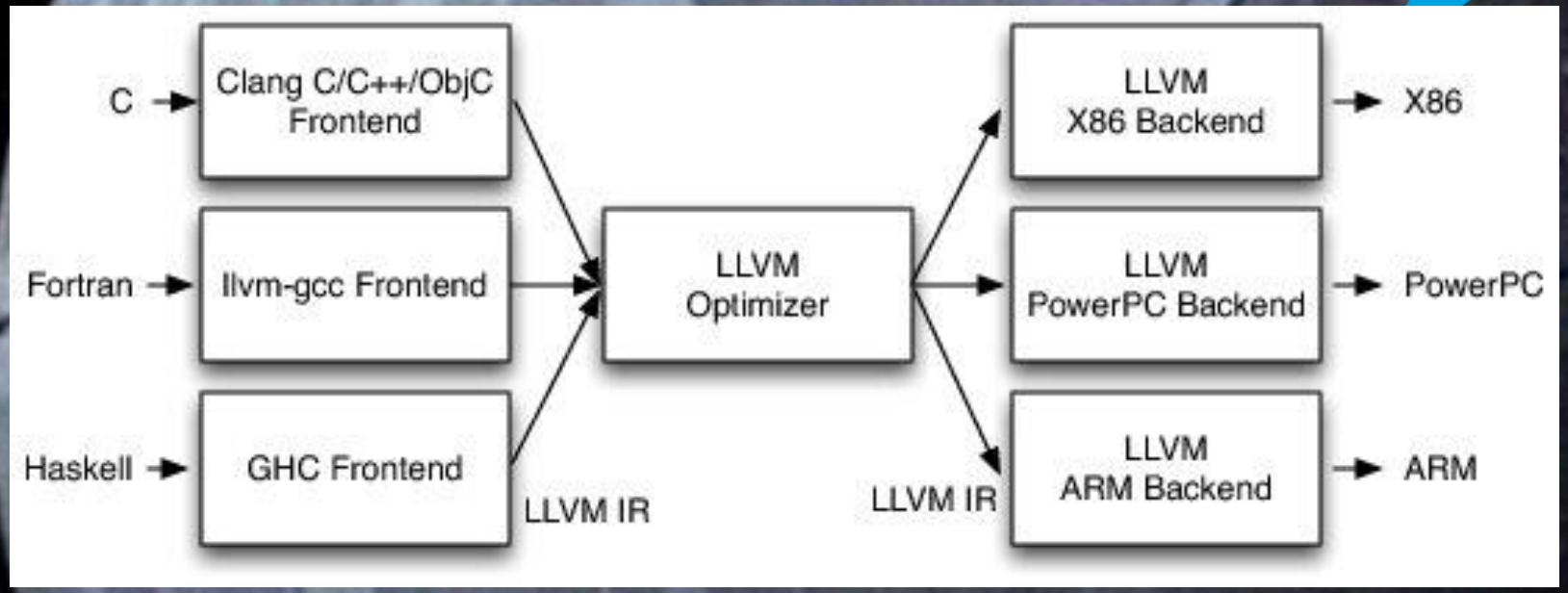
LLVM

tool for create compiler frontend, because it otimizes the backend


```cpp
#include <llvm/IR/LLVMContext.h>
#include <llvm/IR/Module.h>
#include <llvm/IR/IRBuilder.h>
#include <llvm/IR/Verifier.h>
#include <llvm/Support/raw_ostream.h>

int main() {
    llvm::LLVMContext context;
    llvm::Module* module = new llvm::Module("my_module", context);
    llvm::IRBuilder<> builder(context);

    // Create a simple function: int foo() { return 42; }
    llvm::FunctionType *funcType = llvm::FunctionType::get(builder.getInt32Ty(), false);
    llvm::Function *fooFunc = llvm::Function::Create(funcType, llvm::Function::ExternalLinkage, "foo",
module);
    llvm::BasicBlock *entry = llvm::BasicBlock::Create(context, "entry", fooFunc);
    builder.SetInsertPoint(entry);
    builder.CreateRet(builder.getInt32(42));

    // Verify the function
    llvm::verifyFunction(*fooFunc);

    // Print the generated LLVM IR
    module->print(llvm::outs(), nullptr);

    delete module;
    return 0;
}
```

```
; ModuleID = 'my_module'
source_filename =
"my_module"
define i32 @foo() {
entry:
  ret i32 42
}
```

PYLLVM & NUMBA



PUSHDOWN AUTOMATA FOR STACK ARIMETIC

principle of 1 time code , excutable
in all computers

Example [edit]

Consider the following Java code:

```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

A Java compiler might translate the Java code above into bytecode as follows, assuming the above was put in a method:

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic       #84; // Field java/lang/System.out:Ljava/io/PrintStream;
34:  iload_1
35:  invokevirtual   #85; // Method java/io/PrintStream.println:(I)V
38:  iinc    1, 1
41:  goto    2
44:  return
```

Brainfuck [edit]

Brainfuck is designed for extreme minimalism and leads to obfuscated code, with programs containing only eight distinct characters. The following program outputs "Hello, world!":[14]

```
++++++++++[>+++++++>++++++++++>+++<<<-]>++.>+.+++++++
..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.
```

All characters other than `+-<>,.[]` are ignored.

Whitespace [edit]

Whitespace uses only whitespace characters (space, tab, and return), ignoring all other characters.

Piet program that prints 'Piet'

A "Hello World" program in Piet

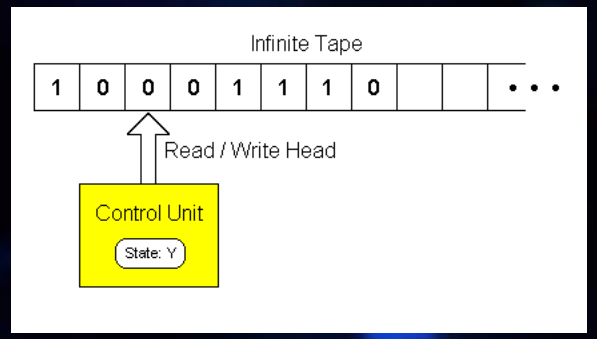
ESOTERIC PROGRAMING LANGUAGES

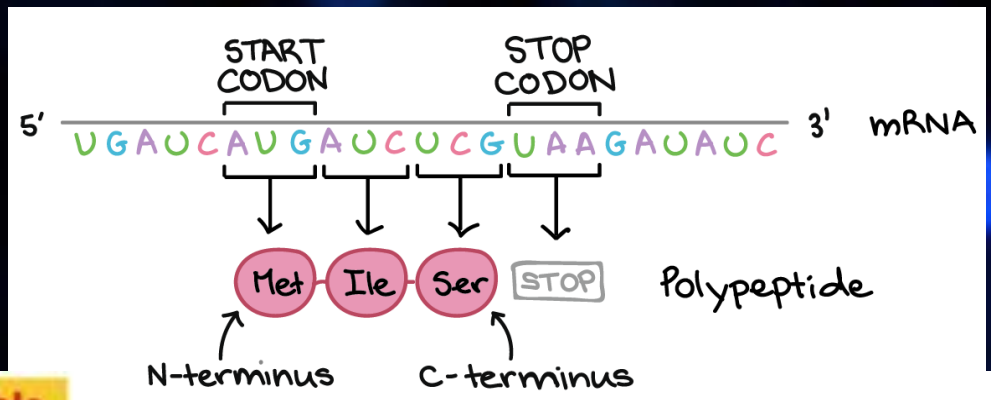
https://esolangs.org/wiki/Main_Page

HelloWorld.dna

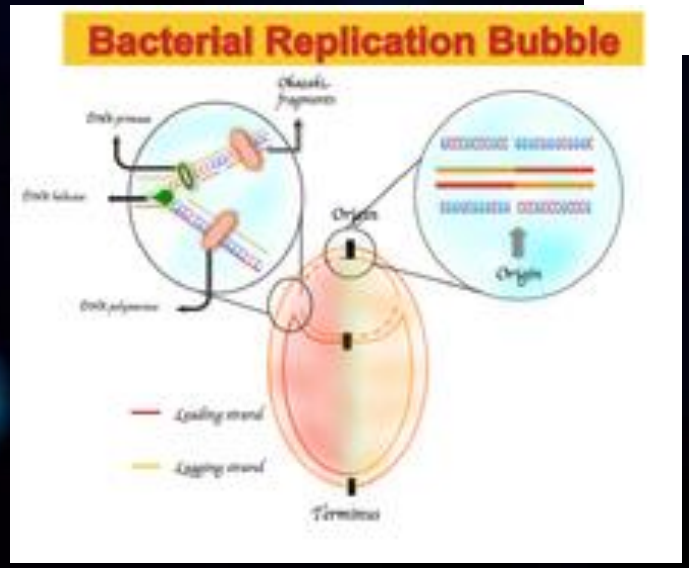
```
AT
T--A
A----T
T-----A
T-----A
G----C
T--A
GC
CG
C--G
A----T
A-----T
T-----A
A----T
A--T
GC
AT
C--G
T----A
C-----G
T-----A
G----C
C--G
CG
AT
A--T
T----A
A-----T
A-----T
G----C
A--T
GC
TA
G--C
T----A
G-----C
C-----G
C----G
A--T
GC
TA
G--C
A----T
G-----C
A-----T
C----G
A--T
CG
```

https://esolangs.org/wiki/RNA#DNA





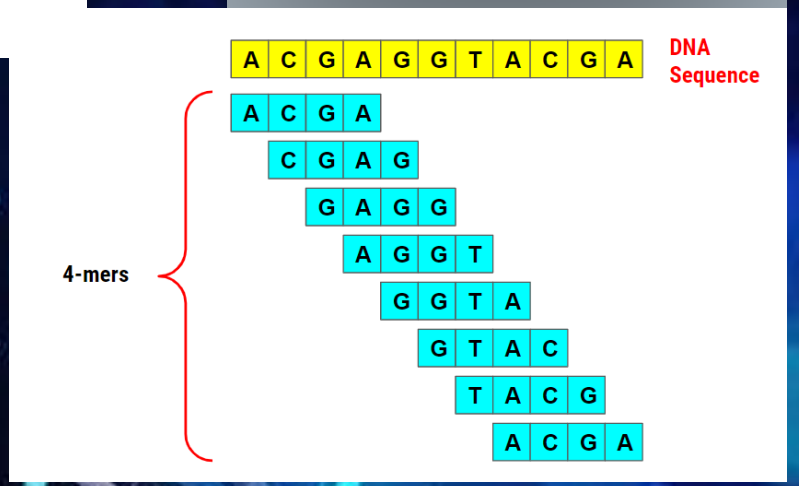

```
>sequence1
ATGCGTACGTAGCTAGCTA
>sequence2
GCTAGCTAGCTAGTACGTA
```



this programing language let you know the replication position of bacteria with:

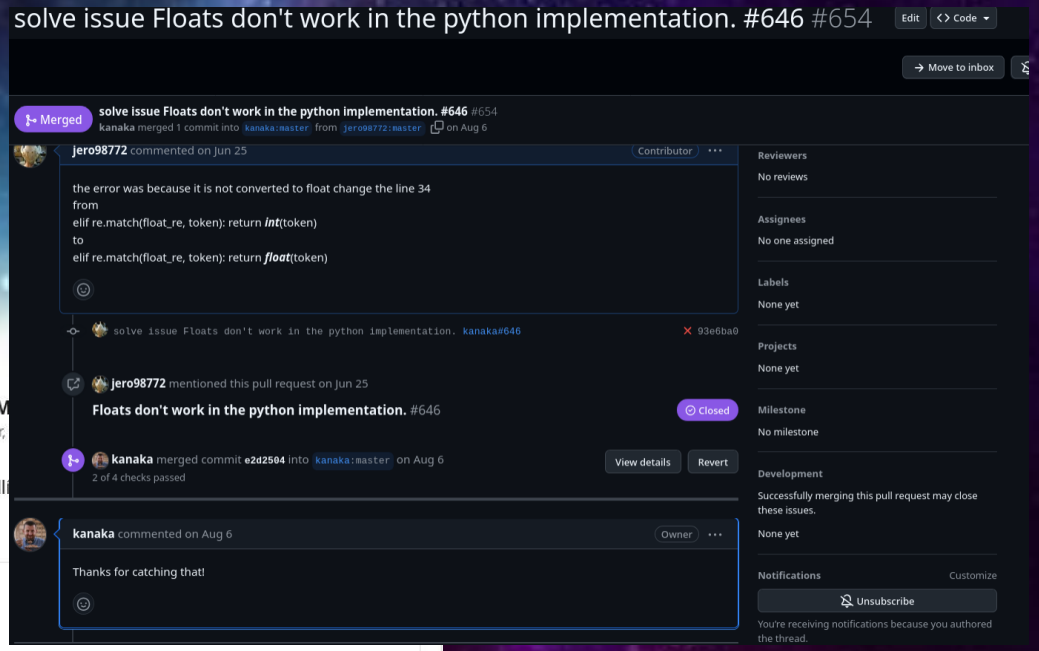
If g +=1

if c -=1



TURING-MACHINE-GENOME-PROCESSOR

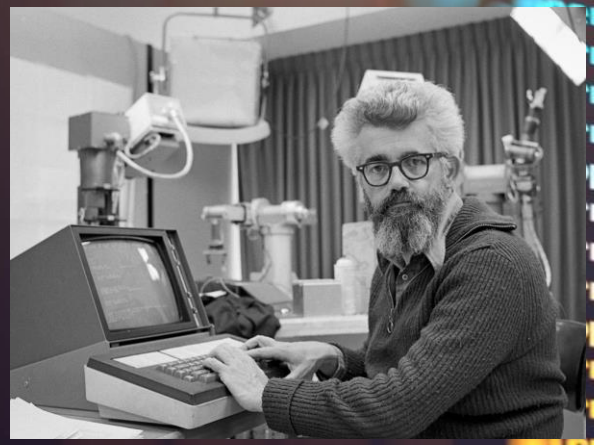
programing language for dna & rna



CONTEXT OF THE IMPORTANCE OF THIS SPEAK


DERUUK A MULTILINGUAL
PROGRAMING LANGUAGE MADE PYTHON
(jero98772)


Lisp dialect in
De-> Deustche Spreche
(german language)
Ru->русский язык
(russian language)
Uk->Українська мова
(ukraine language)
WHAT
IS DERUUK



Recursive Functions of Symbolic Expressions
and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge, Mass. *

April 1960

1 Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions. The original proposal [1] for the Advice Taker was made in November 1958. The main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions.

In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions. This representation is independent of the IBM 704 computer, or of any other electronic computer, and it now seems expedient to expound the system by starting with the class of expressions called S-expressions and the functions called S-functions.

*Putting this paper in LaTeXpartly supported by ARPA (ONR) grant N00014-94-1-0775 to Stanford University where John McCarthy has been since 1962. Copied with minor notational changes from *CACM*, April 1960. If you want the exact typography, look there. Current address, John McCarthy, Computer Science Department, Stanford, CA 94305, (email: jmc@cs.stanford.edu), (URL: http://www-formal.stanford.edu/jmc/)

WHAT IS LISP? NOT A PROGRAMING LANGUAGE


(drucken "hallo"
(eingabe "wie heisst du\n"))
print("hello",input("how is your name"))
(печать "привет Мир")
print("hello word")
HOW PEOPLE SEE PROGRAMERS OF DERUUK

EXAMPLES OF DERUUK

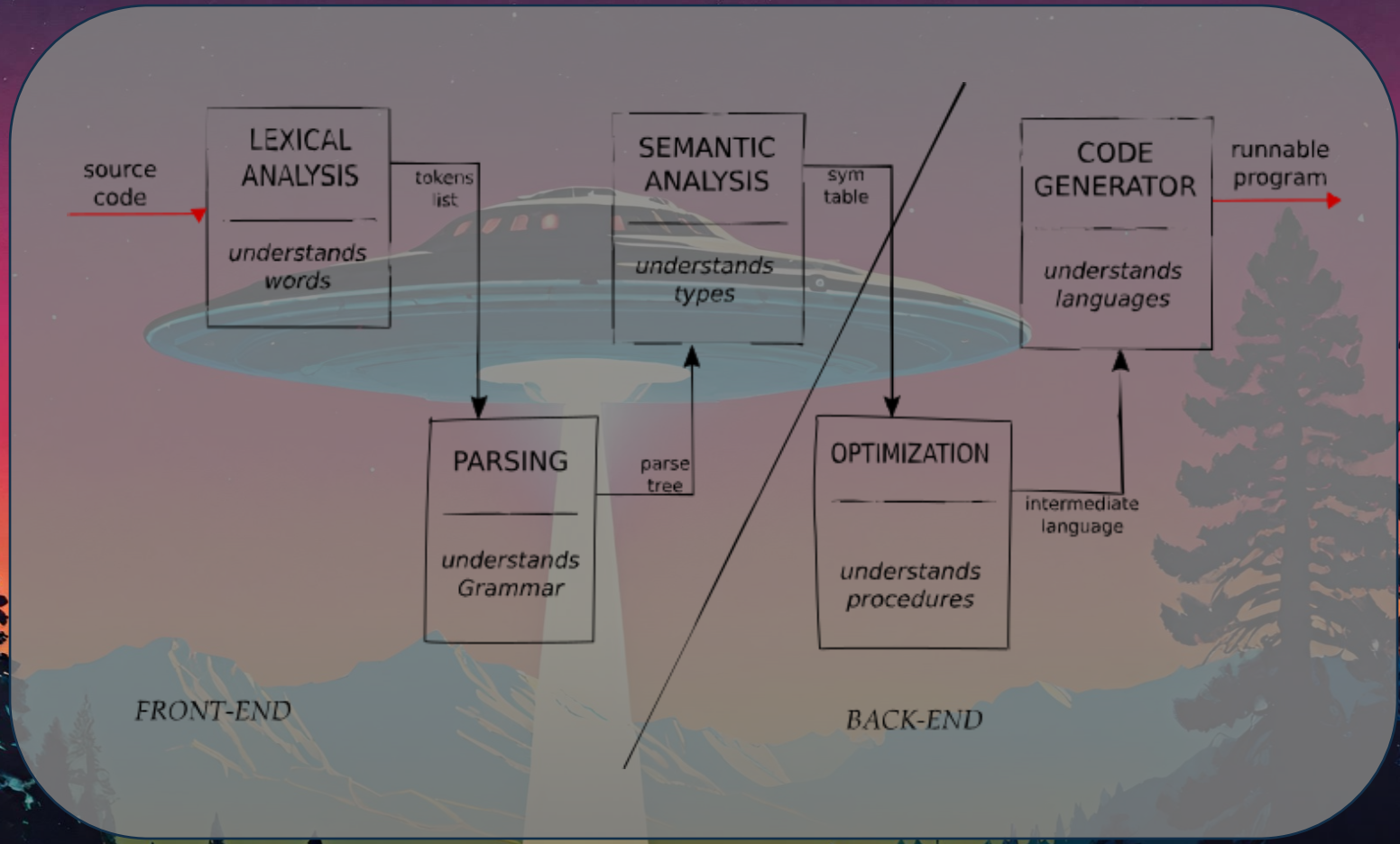
```
(+ (входнойавтомат) (входнойавтомат))

(+ (autoinput) (autoinput))
```

```
(леть sum (фн [a] (/ ( * a (+ a 1) ) 2)))

(sum 100 )


(let sum (fn [a] (/ ( * a (+ a 1) ) 2)))

(sum 100 )
```

```
(wenn (== (% (eingabeautomatik) 2) 0 ) (druckenzl "even") (druckenzl "odd") )


(if (== (% (autoinput) 2) 0 ) (println "even") (println "odd") )
```

HOW CAN I DO A LISP DIALECT IN PYTHON

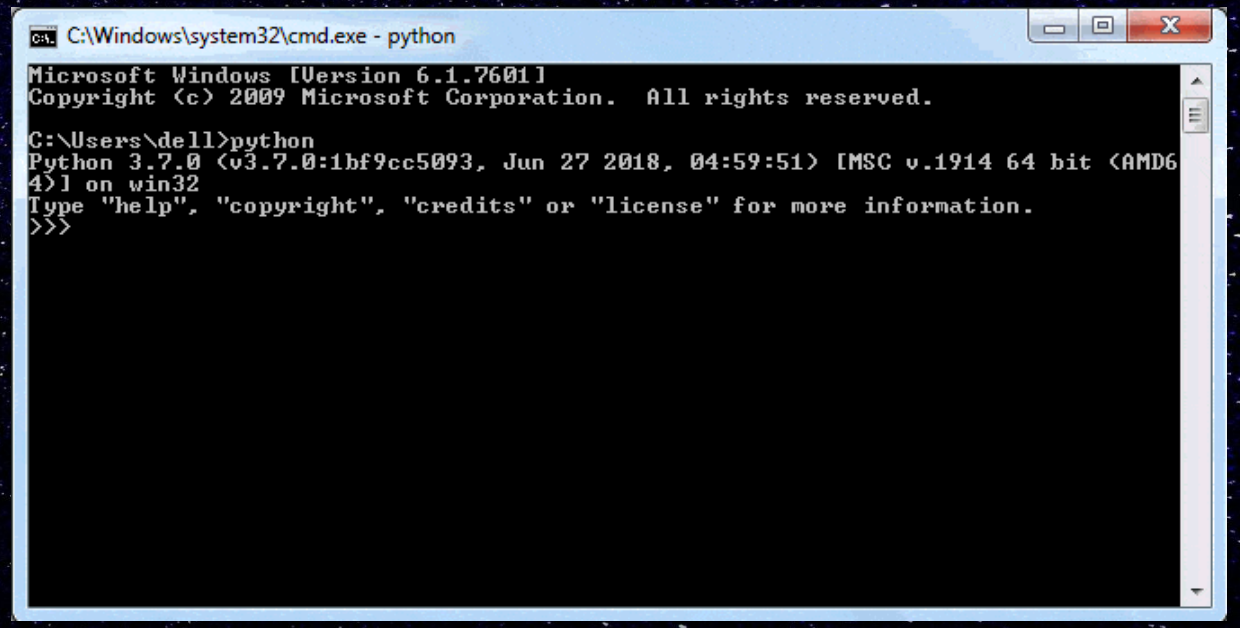
- step1_read_print
- step2_eval ...
- step3_env
- step4_if_fn
- Extra features



https://github.com/kanaka/mal

REPL (READ-EVAL-PRINT LOOP)

```python
def REP(str):
    return PRINT(EVAL(READ(str), {}))

# repl loop
while True:
    REP(input())
```



EVAL

EVAL IS WHERE IS DEFINED CONTROL STRUCTURES THAN CHOOSES A DIRECTION TO GO BASED ON GIVEN PARAMETERS

```
el = eval_ast(ast, env)
f = el[0]
if hasattr(f, '__ast__'):
    ast = f.__ast__
    env = f.__gen_env__(el[1:])
else:
    return f(*el[1:])
```

TOKENIZING

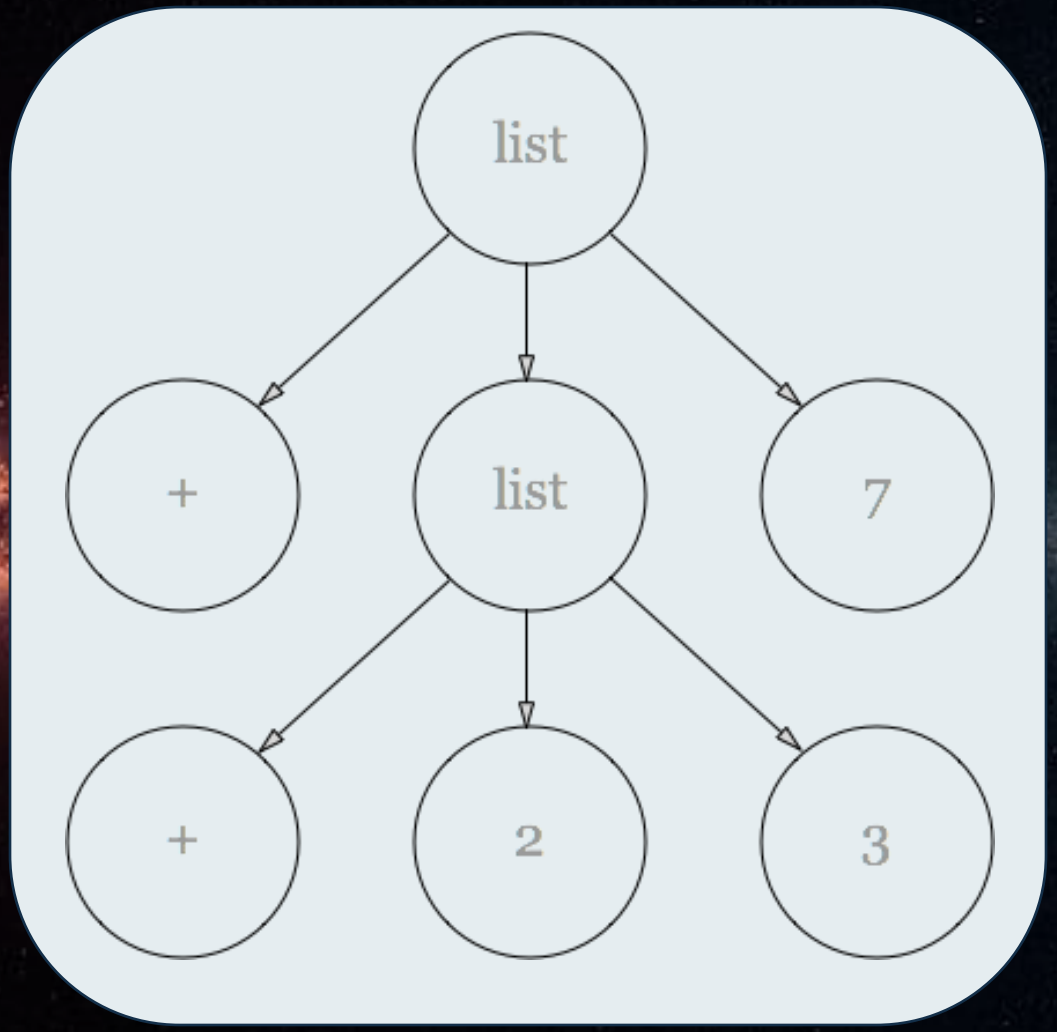
```
def tokenize(str):
    tre = re.compile(r"""[\s,]*(~@|[\[\]{}()'`~^@]|"(?:[\\].|[^\\"])*"?|;.*|[^\s\[\]{}()'"`@,;]+)""");
    return [t for t in re.findall(tre, str) if t[0] != ';']
```

```
'(', 'let', 'quadrat', '(','fn', '[x]', '(','*','x', 'x)',')',')'
```

Procces to separate the string in substrings (called token), but each substring have a meaning , it can be a data type, operator or a list

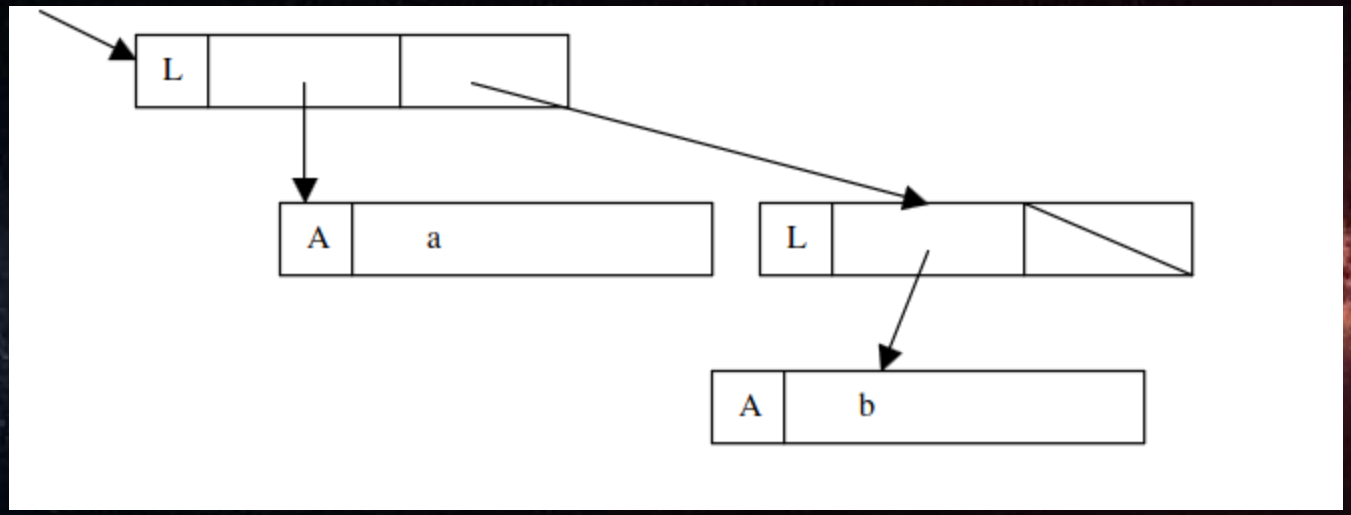
SINTAXCS ANALYS

- the process for verifying that our grammar is being followed



S-EXPRESIONS

There are two kinds of S-expression, atoms and lists.



S-expressions express your program and manipulate it the same way a interpreter would. This lets you add your own language features without modifying the compiler or interpreter.

SEMANTICS

PROCESES OF GIVING
MEAN TO THE SENTENCE

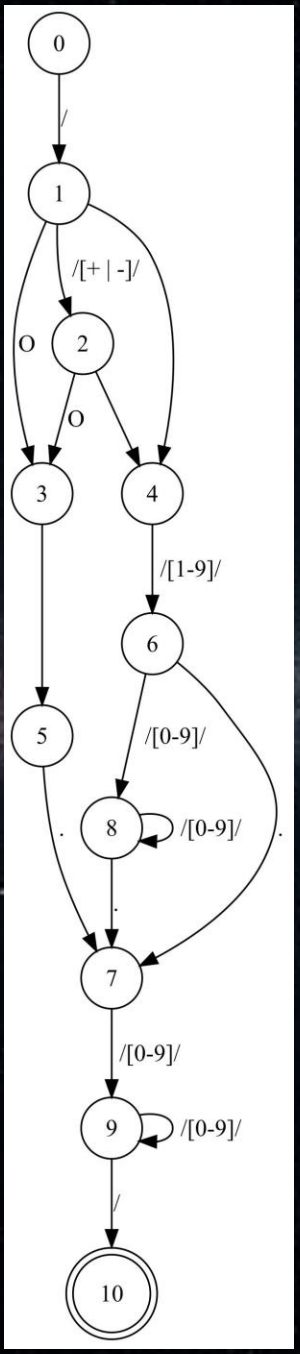
```
NameError: name 'a' is not defined
```

```
TypeError: can only concatenate str (not "int") to
str
```

POWER OF AUTOMATAS

- Modeling Ram
- Grammar Checking
- Speed algorithms
- Lexical Analysis
- Type Checking
- Parsing

```
def read_atom(reader):
    int_re = re.compile(r"-?[0-9]+$")
    float_re = re.compile(r"-?[0-9][0-9.]*$")
    string_re = re.compile(r'"(?:[\\].|[^\\"])*"')
    token = reader.next()
    if re.match(int_re, token):     return int(token)
    elif re.match(float_re, token): return float(token)
    elif re.match(string_re, token):return _s2u(_unescape(token[1:-1]))
```



ENVIROMENT

Symbol Table

```
int count;
char x[] = "NESO ACADEMY";
```

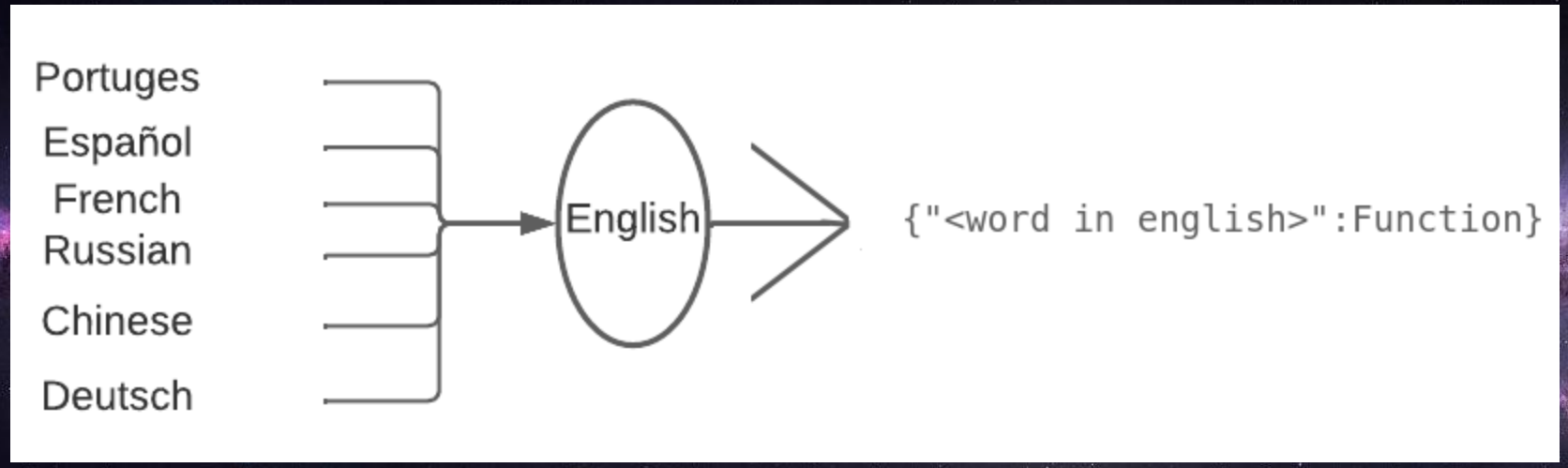
Entries

Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address
count	int	2	0	--	--	--
x	char	12	1	--	--	--

03 Compiler Design

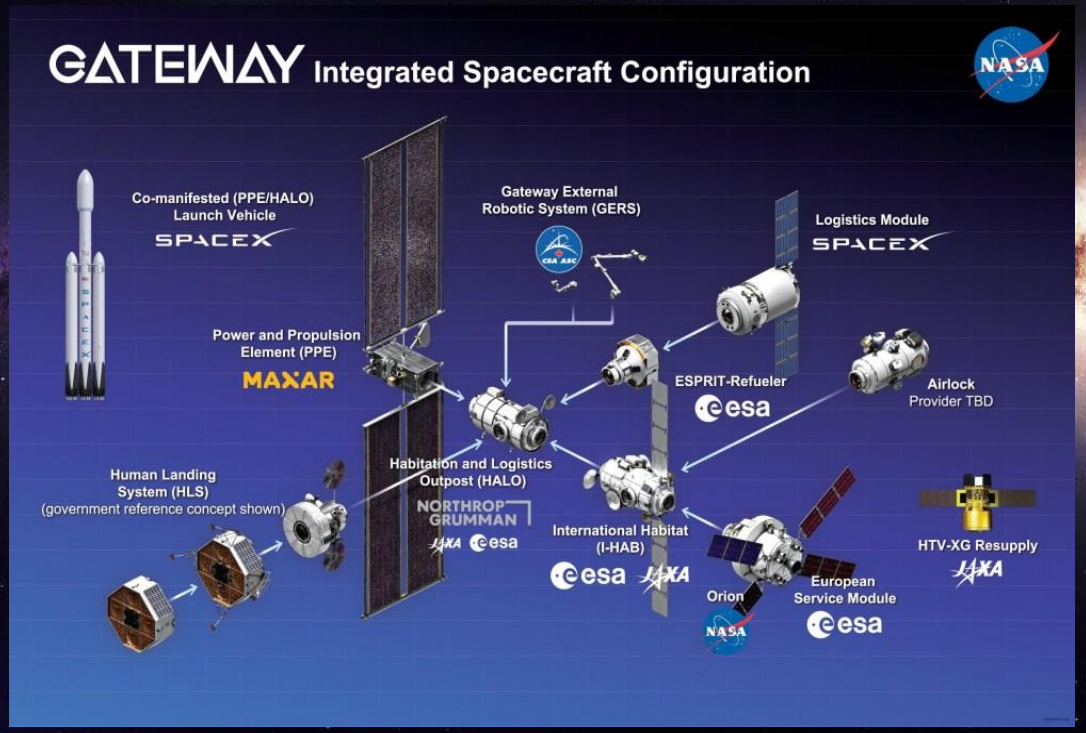
```
env = {
        '==': types._equal_Q,
        '<':  lambda a,b: a<b,
        '<=': lambda a,b: a<=b,
        '>':  lambda a,b: a>b,
        '>=': lambda a,b: a>=b,
        '+':  lambda a,b: a+b,
        '-':  lambda a,b: a-b,
        '*':  lambda a,b: a*b,
        '/':  lambda a,b: int(a/b),
        '%':  lambda a,b: int(a%b),
        'бросок': throw,
        'строка-отображение': pr_str,
        'строка': do_str,
        'печать': prn,
        'вход': input,
        'входнойавтомат': inputeval,
        'werfen': throw,
        'drucken': prn,
        'druckenzl': println,
        'eingabe': input,
        'eingabeautomatik': inputeval
        #... 300 lines of code later
}
```

A PROGRAMING LANGUAGE FOR ALL LANGUAGES?



MACROS

Rule or pattern that specifies how a certain input should be mapped to a replacement output.

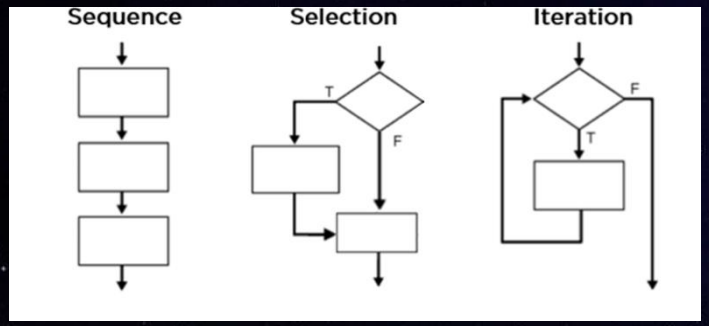



```lisp
(defmacro square (x)
  `(* ,x ,x))
```

```c
#define SQUARE(x) ((x) * (x))
```

python dont have macros but we can create it with eval()

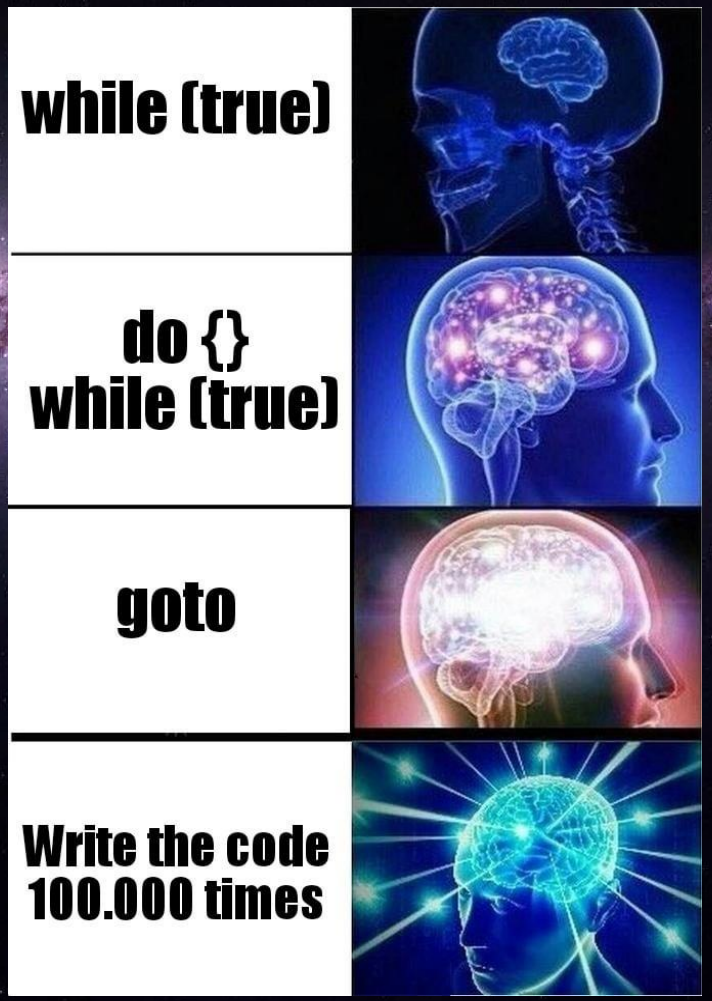
CONTROL STRUCTURES




```
def EVAL(ast, env):
        if not types._list_Q(ast):
            return eval_ast(ast, env)
        if len(ast) == 0: return ast
        a0 = ast[0]

        if "let" == a0 or "леть" == a0:
            a1, a2 = ast[1], ast[2]
            res = EVAL(a2, env)
            return env.set(a1, res)
        elif "def" == a0 or "деф" == a0:
            a1, a2 = ast[1], ast[2]
            let_env = Env(env)
            for i in range(0, len(a1), 2):
                let_env.set(a1[i], EVAL(a1[i+1], let_env))
            return EVAL(a2, let_env)
        elif "do" == a0:
            el = eval_ast(ast[1:], env)
            return el[-1]
        elif "if" == a0 or "wenn" == a0 or "если"  == a0 or "якщо" == a0:
            a1, a2 = ast[1], ast[2]
            cond = EVAL(a1, env)
            if cond is None or cond is False:
                if len(ast) > 3: return EVAL(ast[3], env)
                else:            return None
            else:
                return EVAL(a2, env)
        elif "fn" == a0 or "фн" == a0:
            a1, a2 = ast[1], ast[2]
            return types._function(EVAL, Env, a2, env, a1)
        else:
            el = eval_ast(ast, env)
            f = el[0]
            return f(*el[1:])
```

```
#...
a0 = ast[0]
#...
elif "if" == a0 or "wenn" == a0 or "если"  == a0 or "якщо" == a0:
    a1, a2 = ast[1], ast[2]
    cond = EVAL(a1, env)
    if cond is None or cond is False:
        if len(ast) > 3: return EVAL(ast[3], env)
        else:            return None
    else:
        return EVAL(a2, env)
elif "fn" == a0 or "фн" == a0:
    a1, a2 = ast[1], ast[2]
    return types._function(EVAL, Env, a2, env, a1)
#...
```

WHERE IS THE FOR OR WHILE?



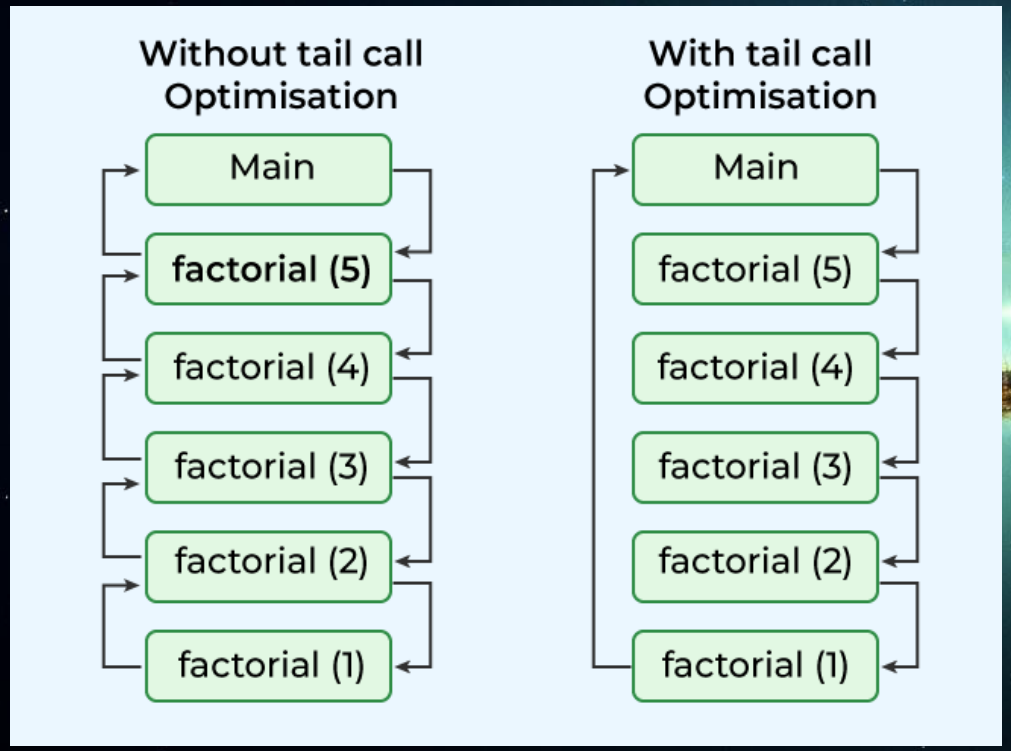
WHERE IS THE FOR OR WHILE?

```
  #just a comment
  """
        elif "while" == a0 or "während" == a0 or "пока"  == a0 or "поки" == a0:
            a1, a2 = ast[1], ast[2]
            while True:  # Infinite loop until condition breaks it
                cond = EVAL(a1, env)
                if cond is None or cond is False:
                    ast = None
                    break
                else:
                    ast = EVAL(a2, env
"""
```


FINALLY WE HAVE A PROGRAMING LANGUAGE

```
(леть factorial  (фн [n] (если (== n 1 ) 1 (* n (factorial (- n 1))) )) )
(factorial 100)
```

TCO (TAIL CALL OPTIMIZATION)



Beneficts:

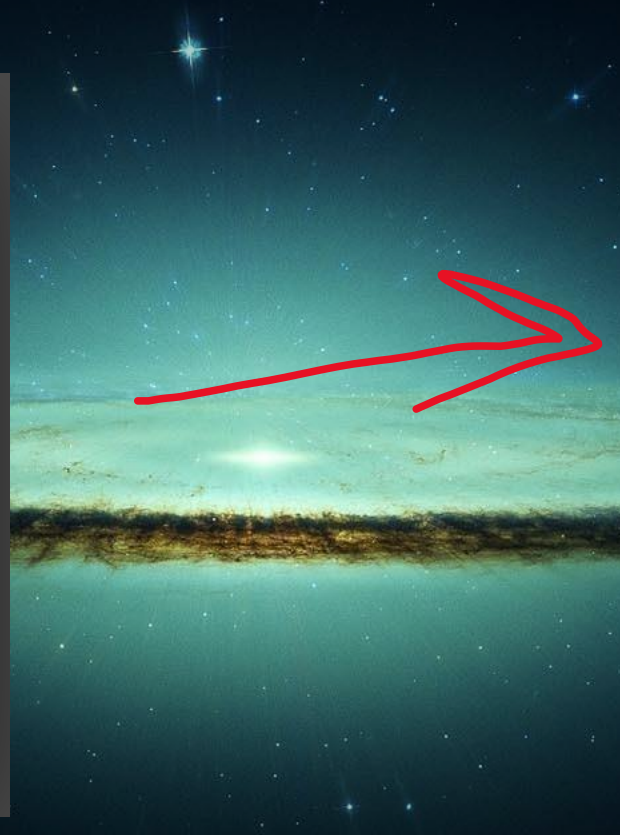
- avoids unnecessary stack manipulation operations.
- No stack overflow
- Improved Performance

TCO (TAIL CALL OPTIMIZATION)

Tail call optimizationin Python is not natively supported due to Python's recursion limit and the lack of tail call optimization in its interpreter. However, it can be simulated using a loop to replace recursive calls with iterative ones.

"TCO (TAIL CALL OPTIMIZATION)" SIUMALTED

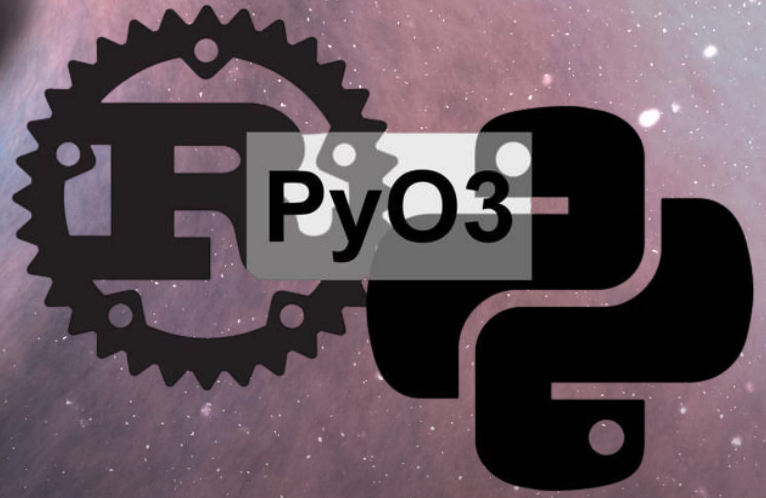
```python
def EVAL(ast, env):
        #print("EVAL %s" % printer._pr_str(ast))
        if not types._list_Q(ast):
            return eval_ast(ast, env)

        # apply list
        if len(ast) == 0: return ast
        a0 = ast[0]

        if "let" == a0 or "леть" == a0 == a0:
            a1, a2 = ast[1], ast[2]
            res = EVAL(a2, env)
            return env.set(a1, res)
        #...

        else:
            el = eval_ast(ast, env)
            f = el[0]
            return f(*el[1:])
```



```python
def EVAL(ast, env):
        #print("EVAL %s" % printer._pr_str(ast))
    while True:
        if not types._list_Q(ast):
            return eval_ast(ast, env)

        # apply list
        if len(ast) == 0: return ast
        a0 = ast[0]

        if "let" == a0 or "леть" == a0 == a0:
            a1, a2 = ast[1], ast[2]
            res = EVAL(a2, env)
            return env.set(a1, res)
        #...

        else:
            el = eval_ast(ast, env)
            f = el[0]
            if hasattr(f, '__ast__'):
                ast = f.__ast__
                env = f.__gen_env__(el[1:])
            else:
                return f(*el[1:])
```

¿HOW TO ARCHIVE REAL TCO?

ctypes



EXTRA FEATURES

TRY CATCH

```
        elif "versuch" == a0 or "проба" == a0 or "спроба" == a0:
            if len(ast) < 3:
                return EVAL(ast[1], env)
            a1, a2 = ast[1], ast[2]
            if  a2[0] == "fangen" or a2[0] == "поймать" or a2[0] == "зловити":
                err = None
                try:
                    return EVAL(a1, env)
                except types.MalException as exc:
                    err = exc.object
                except Exception as exc:
                    err = exc.args[0]
                catch_env = Env(env, [a2[1]], [err])
                return EVAL(a2[2], catch_env)
            else:
                return EVAL(a1, env);
```

EXTRA FEATURES

QUOTES AND COMMENTS


```
elif "quote" == a0 or "Zitat" == a0 or "цитата" == a0 :
    return ast[1]
```

COMMUN PROBLEMS WHEN WE MADE A PROGRAMING LANGUAGE


FINALLY WE HAVE A PROGRAMING LANGUE


END
Follow me as jero98772
https://github.com/jero98772/deruuk/