

# DERUUK A MULTILINGUAL PROGRAMMING LANGUAGE MADE PYTHON

---

(jero98772)

A close-up photograph of two large-headed alien creatures. They have light brown skin, large black almond-shaped eyes with pinkish-purple eyelids, and a small, thin mouth. Their heads are disproportionately large compared to their bodies. The background is a soft purple.

Lisp dialect in

De-> Deustche Spreche

(german language)

Ru->русский язык

(russian language)

Uk->Українська мова

(ukraine language)

WHAT  
IS DERUUK



```
object to mirror
mirror_mod.mirror_object
operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True
```

# WHAT IS LISP? NOT A PROGRAMMING LANGUAGE

```
selection at the end - add
mirror_ob.select= 1
mirror_ob.select=1
context.scene.objects.active
("Selected" + str(modifier))
mirror_ob.select = 0
bpy.context.selected_objects
data.objects[one.name].select
print("please select exactly one object")
- OPERATOR CLASSES ---
```

```
does Operator does not
    > mirror to the selected
    > object mirror_mirrored
    > for X
        > is not
```

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 expand the system by starting with the class of expressions called S-expressions and the functions called S-functions.

\*Putting this paper in L<sup>T</sup>PX partly 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/>)



(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)
```

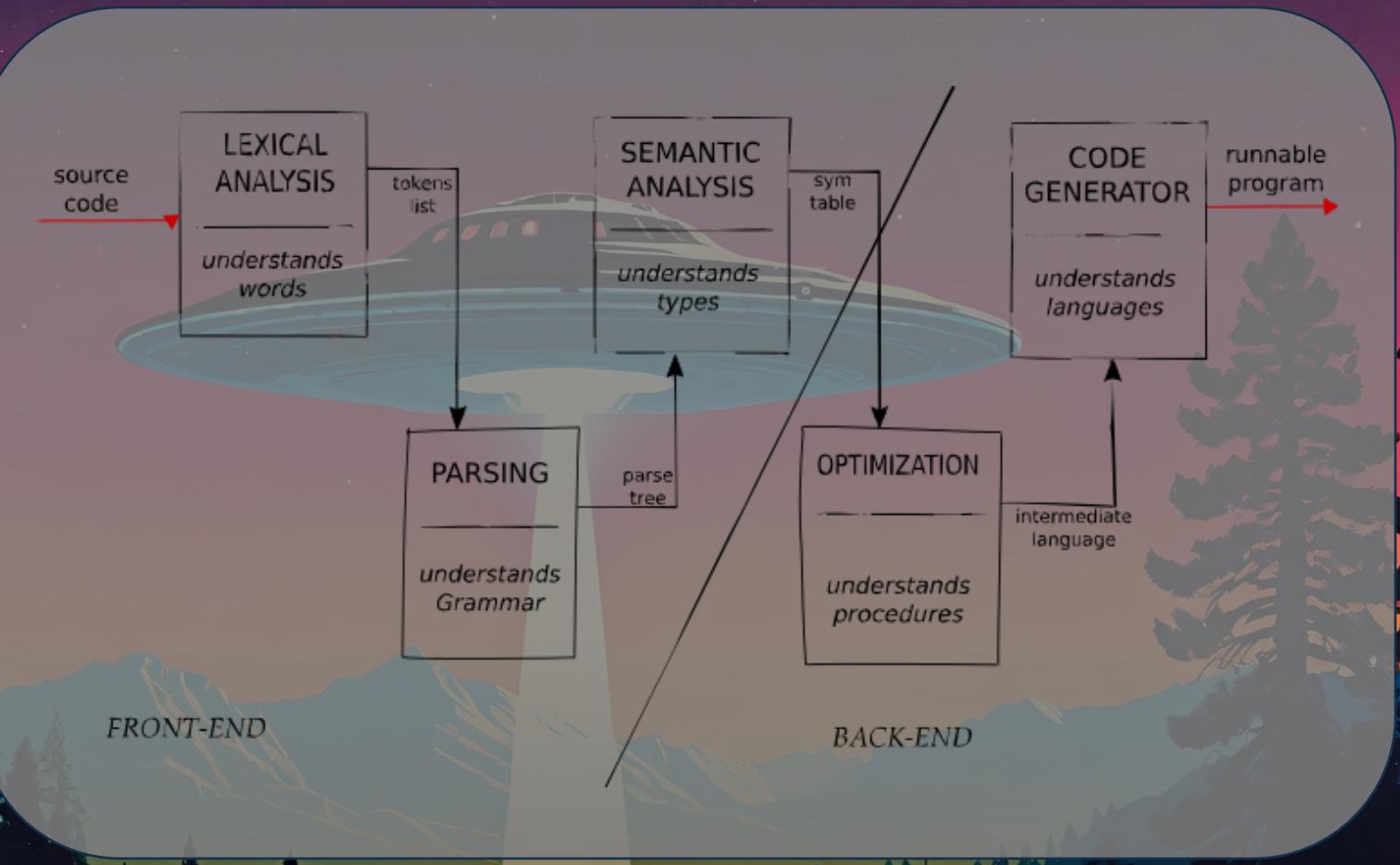


```
(wenn (== (% (eingabeautomatik) 2) 0) (druckenzl "even") (druckenzl "odd"))
```

```
(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

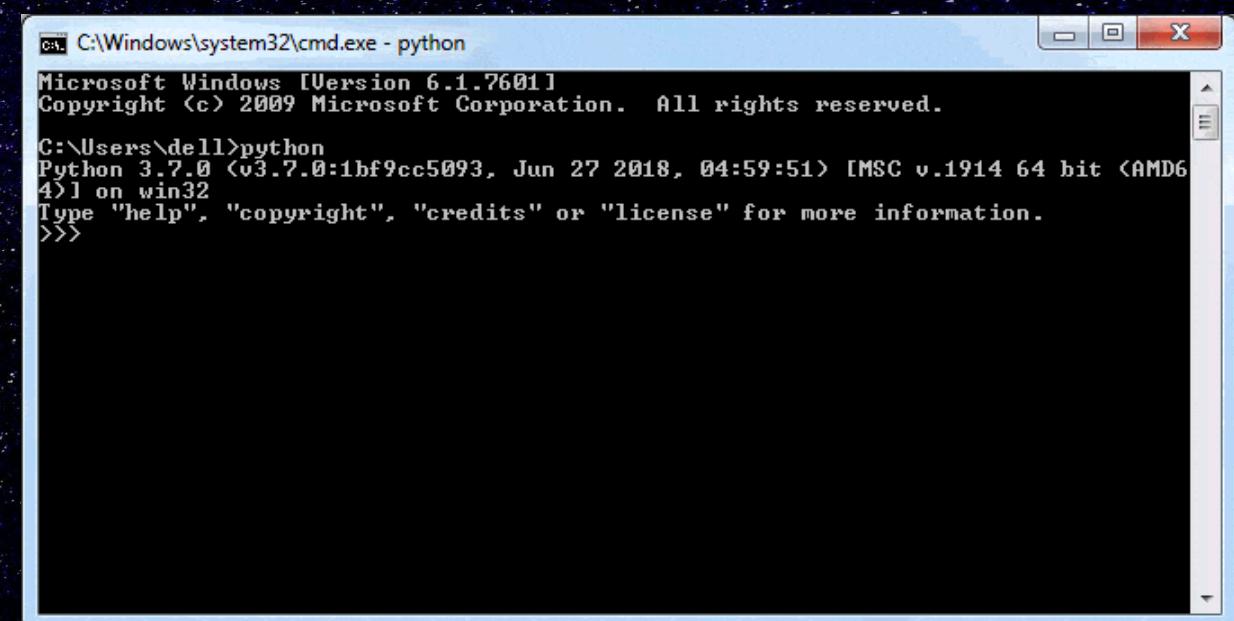


<https://github.com/kanaka/mal>

# REPL (READ-EVAL-PRINT LOOP)

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

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



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - python". The window displays the following text:

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# EVAL

EVAL IS WHERE IS DEFINED  
CONTROL STRUCTURES THAN  
CHOOSSES 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] != ';']
```

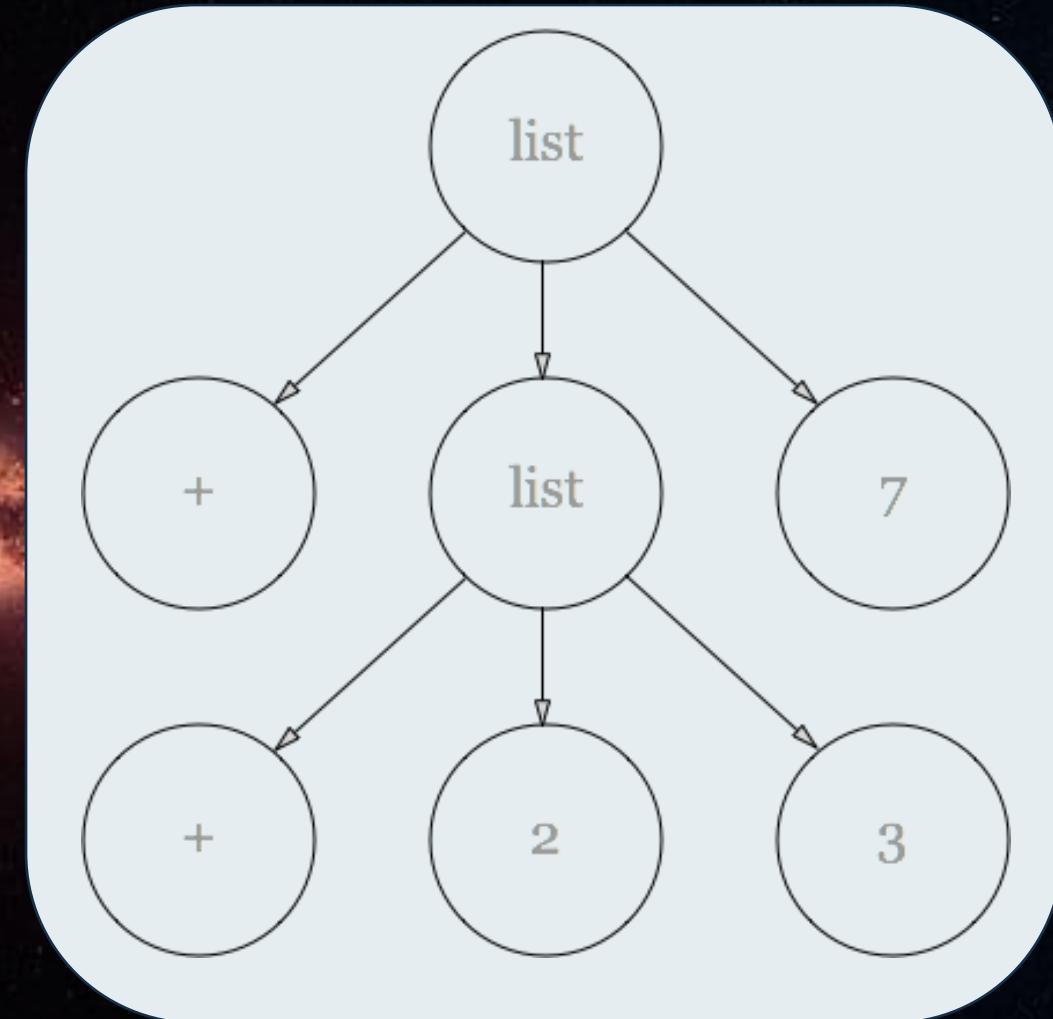
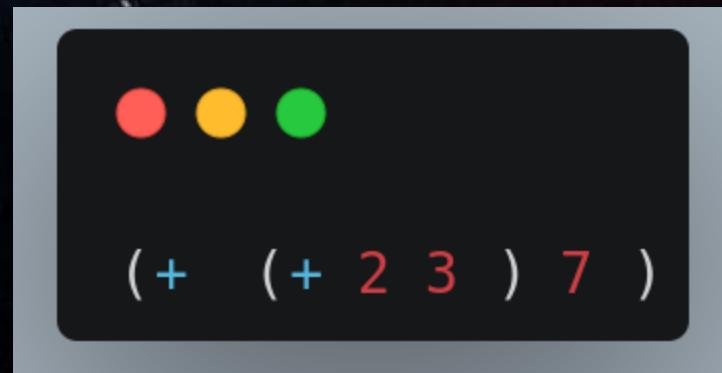


```
'(', 'let', 'quadrat', '(', 'fn', '[x]', '(*', 'x', 'x)', ')', ')'
```

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

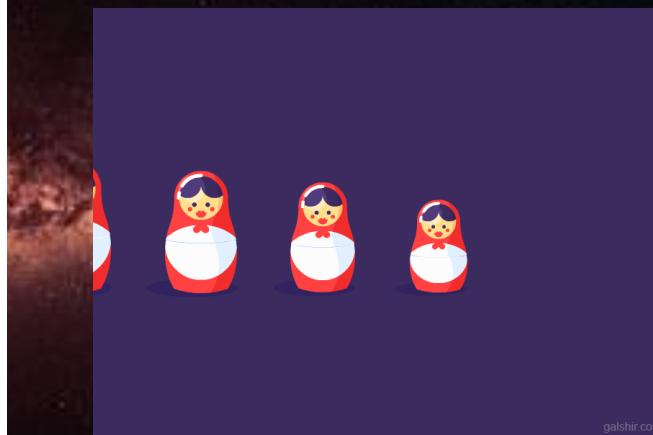
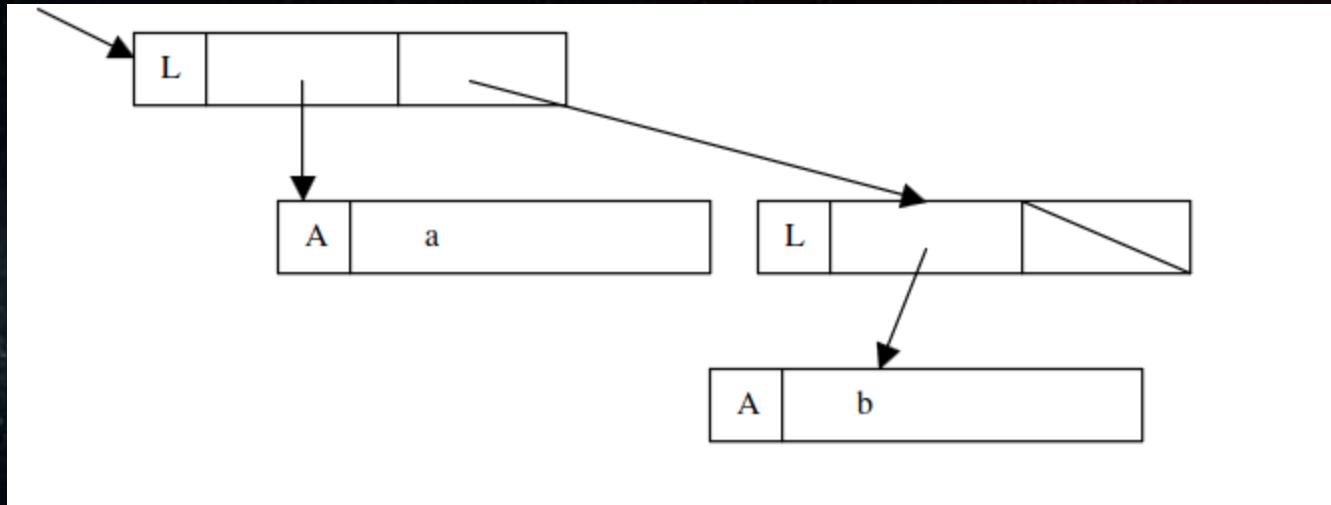
# SINTAXCS ANALYS

- the process for verifying that our grammar is being followed



# S-EXPRESSIONS

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



galshir.com

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

PROCESSES OF GIVING  
MEAN TO THE SENTENCE



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



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

# POWER OF AUTOMATAS

Modeling Ram

Grammar Checking

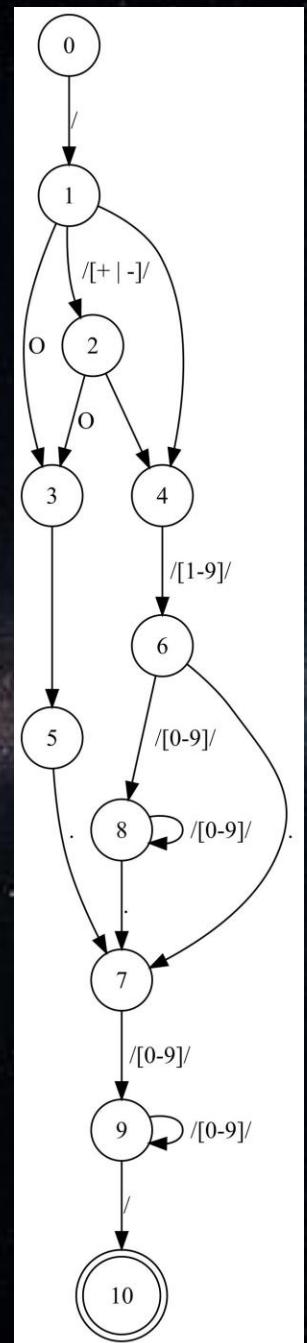
Speed algorithms

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]))
```



# ENVIRONMENT

## Symbol Table

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

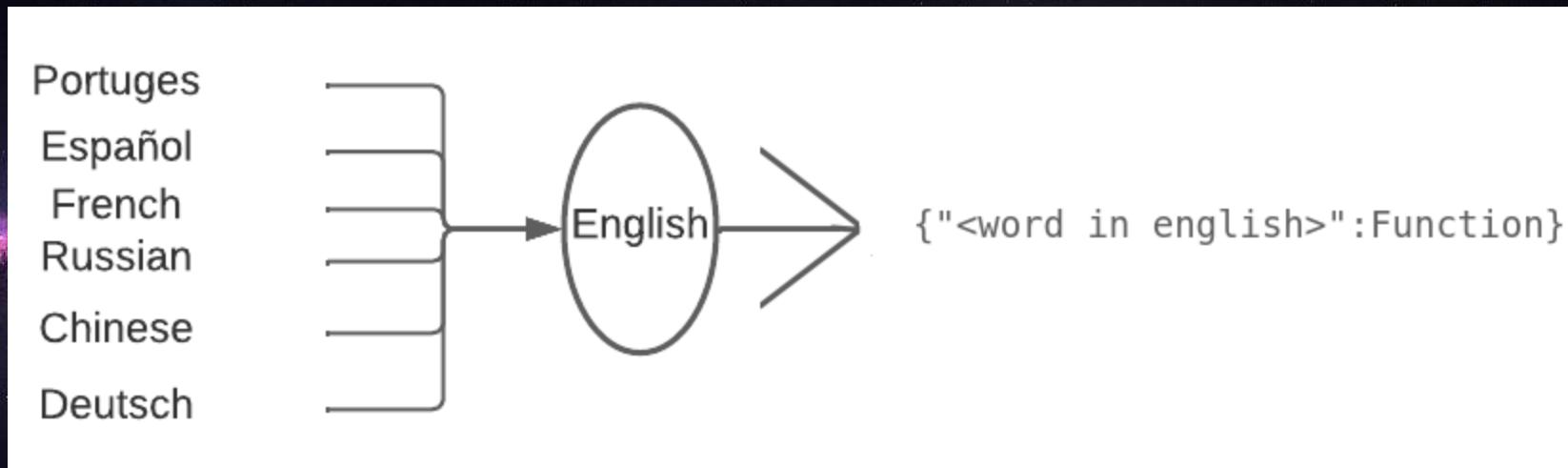
Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address
count	int	2	0	--	--	--
x	char	12	1	--	--	--

Entries

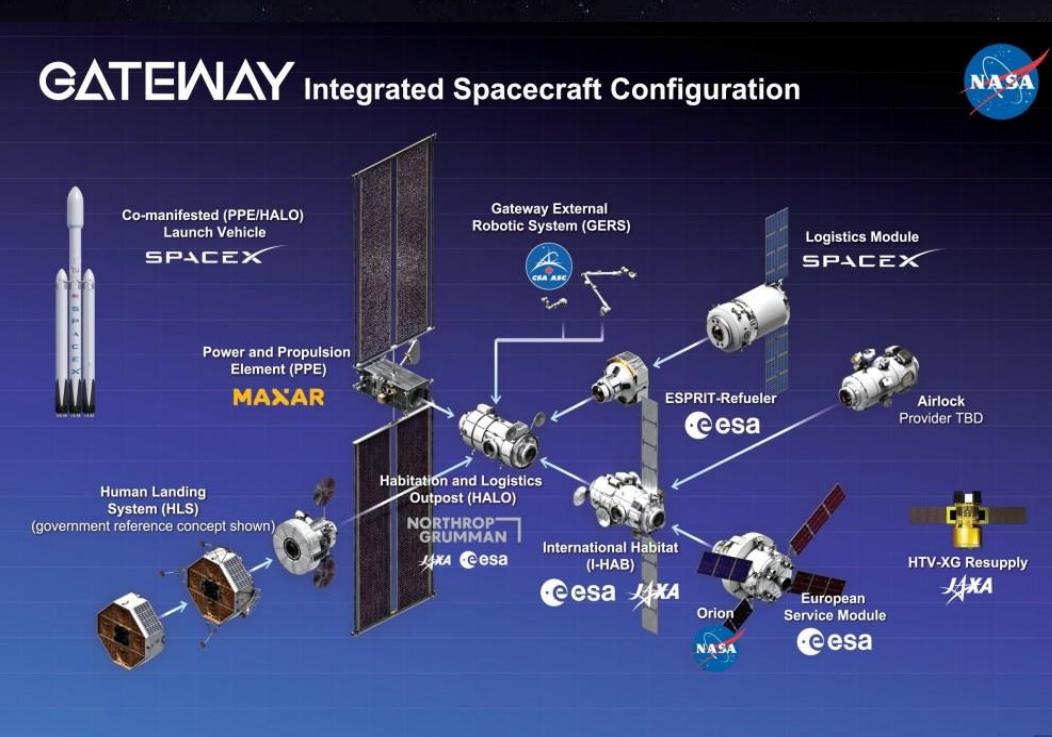
03 / Compiler Design

```
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': inpuEval,  
    'werfen': throw,  
    'drucken': prn,  
    'druckenl': println,  
    'eingabe': input,  
    'eingabeautomatik': inpuEval  
}  
#... 300 lines of code later
```

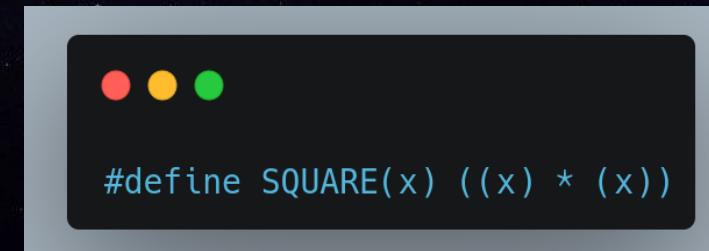
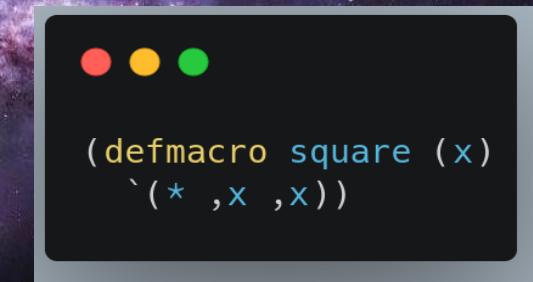
# A PROGRAMMING LANGUAGE FOR ALL LANGUAGES?



# MACROS

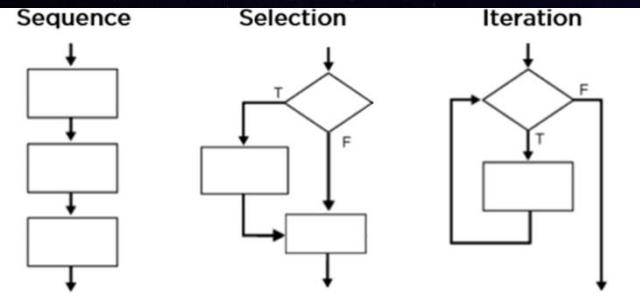


Rule or pattern that specifies how a certain input should be mapped to a replacement output.



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

# CONTROL STRUCTURES

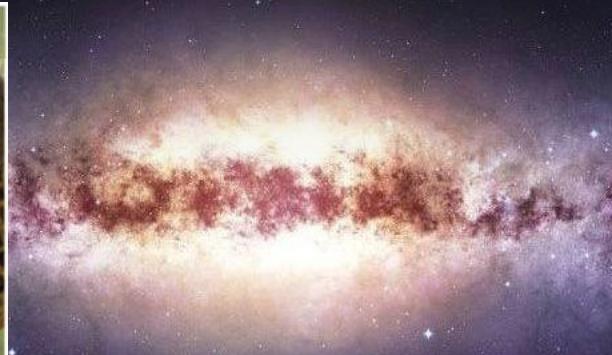


```
def EVAL(ast, env):
    if not types._list_Q(ast):
        return eval_ast(ast, env)
    if len(ast) == 0: return ast
    a0 = ast[0]

    if "let" == a0 or "льть" == 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 "венн" == 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 "венн" == 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?

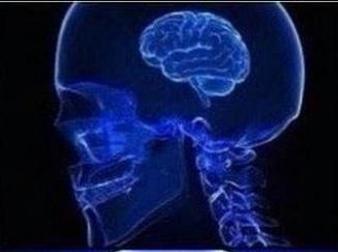


**while (true)**

**do {}  
while (true)**

**goto**

**Write the code  
100.000 times**



# WHERE IS THE FOR OR WHILE?

A wide-angle, low-angle shot of a vast, desolate alien landscape under a dark, cloudy sky. A large, circular, futuristic flying saucer with glowing orange energy rings hovers prominently in the center. In the background, a bright, yellowish-orange sun or moon rises over distant, jagged mountain peaks. The foreground consists of dark, rocky, and craggy terrain.

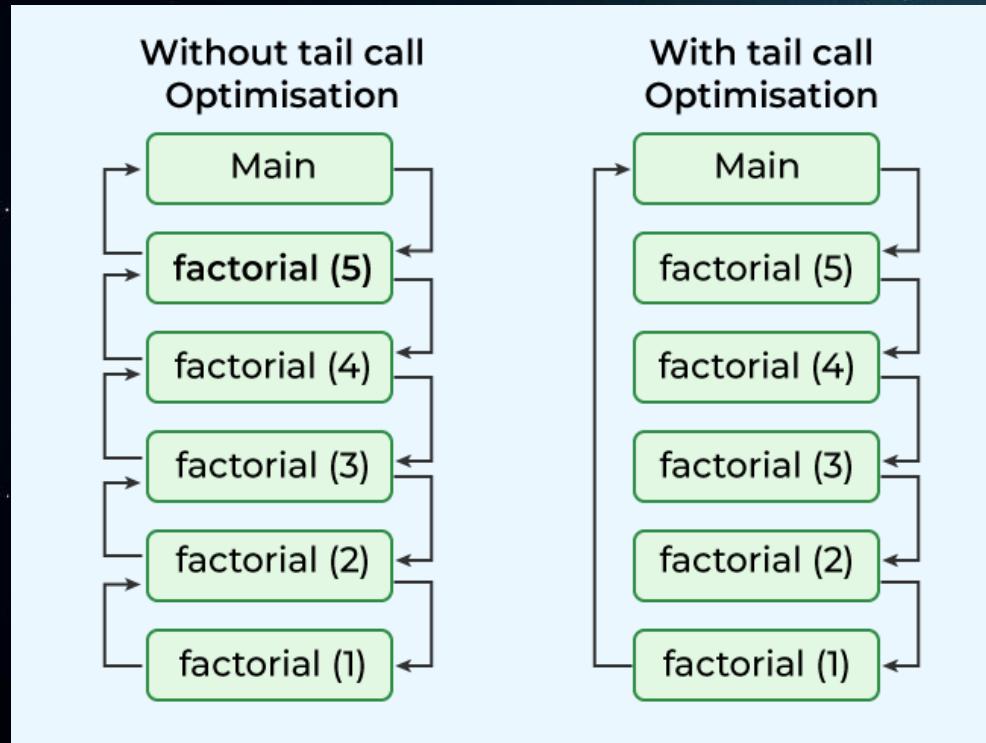
FINALLY WE HAVE A PROGRAMMING LANGUAGE



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



# TCO (TAIL CALL OPTIMIZATION)



Benefits:

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

# TCO (TAIL CALL OPTIMIZATION)

Tail call optimization in 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)" SİUMALTED



```
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 "letъ" == 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:])
```

```
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]

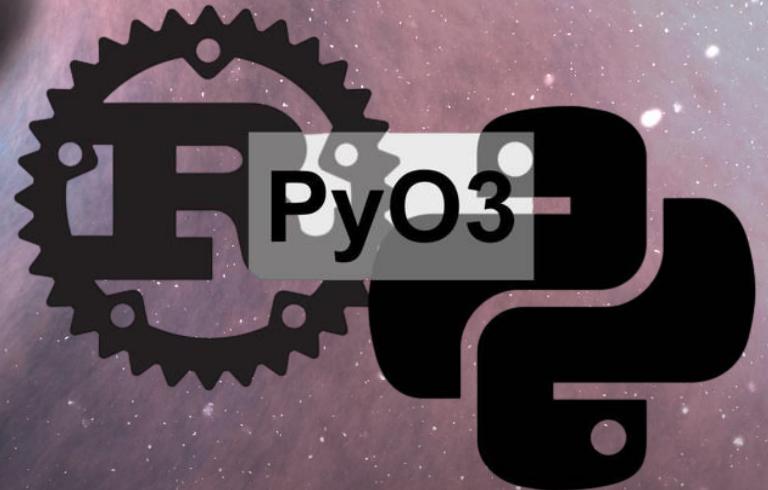
        if "let" == a0 or "letъ" == a0 == a0:
            a1, a2 = ast[1], ast[2]
            res = EVAL(a2, env)
            return env.set(a1, res)
        #...

        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]
```

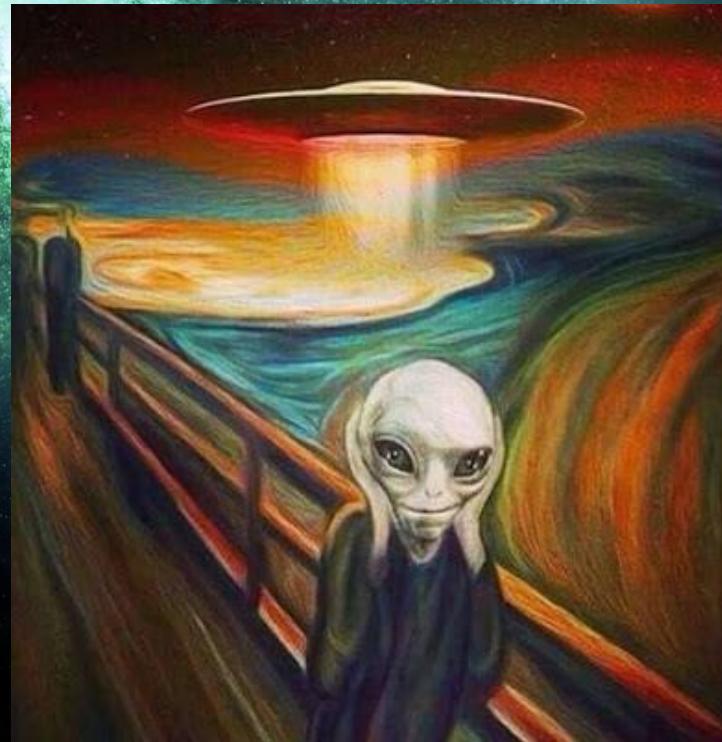
Deruuk IDE

```
1 (let j 98772)  
2 ;(let j 98779)  
3 (quote j)  
4 (quote (+ j j))  
5 (+ j j)  
6 ;made with FastAPI
```

Run Code

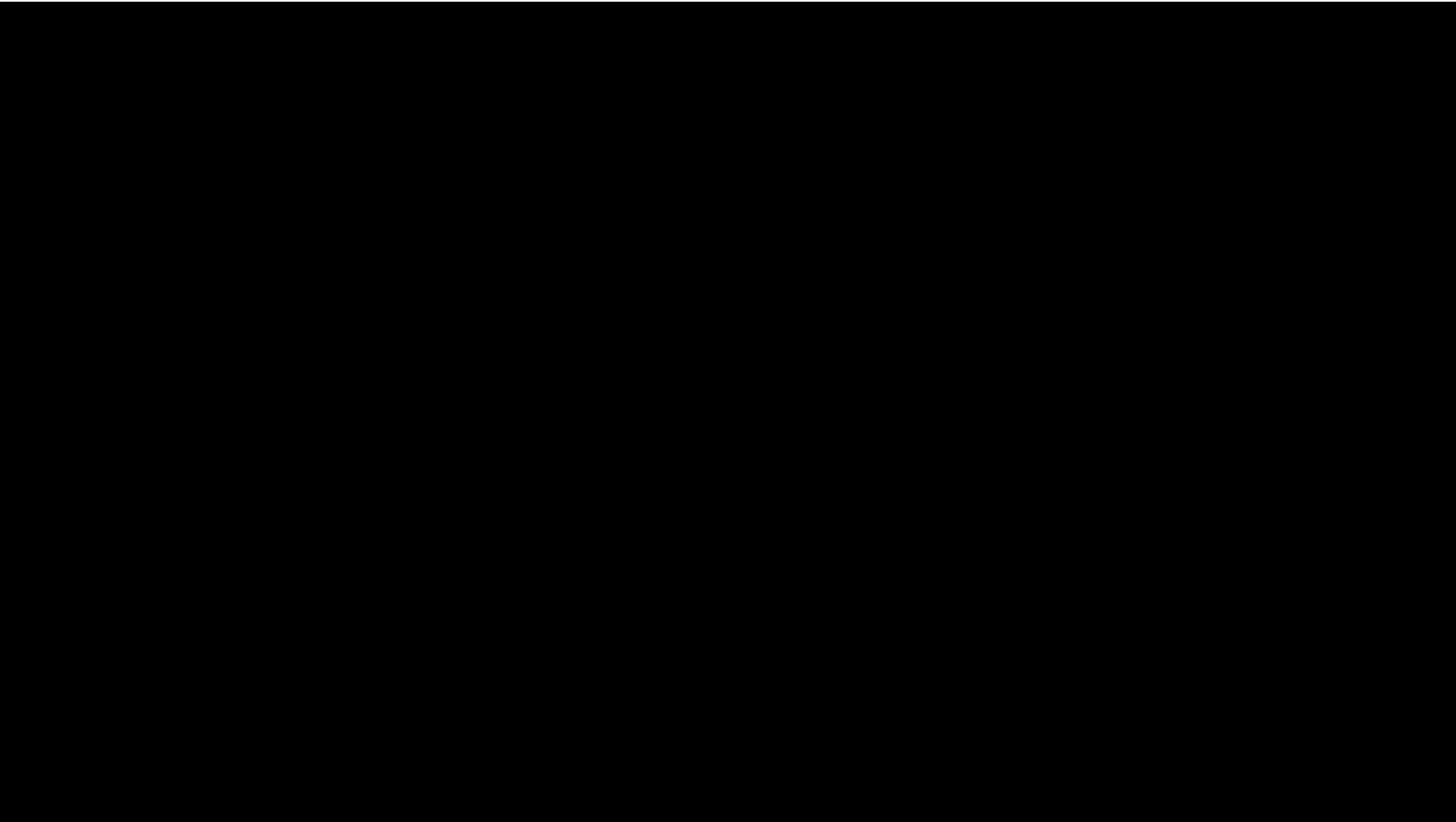
```
98772  
nil  
j  
(+jj)  
197544  
nil
```

# COMMON PROBLEMS WHEN WE MADE A PROGRAMMING LANGUAGE



A wide-angle, low-angle shot of a vast, arid landscape under a dark, cloudy sky. A massive, multi-layered flying saucer or alien mothership hovers in the center, its bottom layer glowing with a bright orange light. To its left, a large, pale sun rises over distant, jagged mountain peaks. In the foreground, the dark, rocky terrain is scattered with smaller, spiky rock formations. The overall atmosphere is one of a science-fiction vision of an alien world.

FINALLY WE HAVE A PROGRAMMING LANGUAGE





END



Follow me as jero98772

<https://github.com/jero98772/deruuk/>